Imail: A DBMS for Electronic Mail

by

KATRYN B. INKLEY

B. Phil, Miami University, 1979
Oxford, Ohio

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

Approved by:

R H Van Brink
Major Professor

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Imail: A DBMS for Electronic Mail

1. Introduction

Electronic mail is an area that is advancing rapidly in the field of computer science. In 1984, the International Telegraph and Telephone Consultative Committee (CCITT) published a set of recommendations that established the standards upon which most new electronic mail systems have been based. The recommendations specified a minimum level of uniformity among systems and serve as a framework upon which enhancements to a basic mail service have been based.

One of these enhancements is incorporating a database management system into electronic mail. One such system, called "imail", was developed to assist a user in organizing and viewing his mail messages. The main focus of this paper is on the merits and implementation of imail. Prior to this, electronic mail is discussed in general so the reader will be familiar with the work being done in this area.

This paper is organized into four chapters. The introduction highlights the main issues. Chapter two is a review of literature that discusses the standards of electronic mail and a variety of systems using those

standards. The issues concerning the development of imail are discussed in chapter three. The final chapter reviews enhancements to imail, and future work with databases and electronic mail.

2. Literature Review

2.1 Introduction

The focus of this project was to integrate a database management system (DBMS) with electronic mail. Although the details of the project will be discussed later, a few items should be noted here. The system that was developed (called "imail") is an add-on to an already existing mail system. It aids the user in processing mail messages on the receiving end by storing them in a database and allowing queries on them. The database used is INGRES* and the mail system is the standard mail facility that is provided with the Unix+ operating system.

A DBMS, in general, is a set of tools that assists users in managing different collections of data. Before DBMSs became popular, all users maintained their own sets of data and were responsible for keeping them up to date. This led to much duplicated data as each user stored all the data that might be needed for a task. For example, an employee's telephone number may

---

* INGRES is a product of Relational Technology, Inc. (RTI)

+ UNIX is a registered trademark of AT&T Bell Laboratories

have been stored three different places in one "database". Inconsistencies arose when the telephone number changed, but was updated in only one or two of the data files. A good DBMS eliminates this data duplication while allowing each user to see the full set of data that is needed to do a specific job. If it is properly laid out, a database can contain information for more than one user with more than one purpose. Each user sees only the data that is needed for the task at hand, and can be assured that it is up to date with the rest of the data in the database.

INGRES is one of the many DBMSs that is available commercially. It is based upon a relational database model and allows a user to easily conceptualize how to retrieve or update all the data related to the current task. Chapter three justifies the use of a DBMS and the selection of the INGRES DBMS.

The rest of this chapter discusses the concept of electronic mail, its current standards in the industry, and some of the research being done to extend its functionality.

2.2 The development of electronic mail

This author believes that electronic mail had very humble origins. It probably began with two people who

worked on the same project, but had somewhat different working hours. They found they frequently needed to leave messages for the other and preferred to do so on-line rather than with notes taped to the face of the other's terminal. And so they agreed on a file name and each would check for messages in the file when logging in. But it became a nuisance to check the file when there were no messages, so one of them wrote a small routine that was automatically invoked when logging in. All it did was write a message to the terminal if the file existed. Thus electronic mail was born.

As the idea spread, more and more people began to do similar tasks. Each set of people had their own file names and some of the routines to check the files were more elaborate than others. But the concept was there, and it was that the computer could be used as a mailbox, an electronic mailbox, for its users.

It soon became apparent that if everyone used a common naming convention for the files, one routine (or a set of routines) could be written to check and read the files for each individual user. Another set of routines could assist the user in writing mail so that each person did not need to know the naming convention for the files. These routines themselves became more

and more complex to allow the user greater flexibility in writing and reading the mail messages.

Suppose now that the members of two different organizations, or perhaps corporations, wished to send mail to one another. Each had their own routines and conventions for sending mail among themselves. But if they tried to send mail to each other, the receiving system might not have known how to interpret the message that was built by the originator. And so there quickly became a need for standards across the industry that directed the format, but not the content, of electronic messages.

ORIGINATING END	RECEIVING END
+ Usually people	+ Usually people
+ Holds message until recipient system is available	+ Stores message until recipient chooses to process it
+ Aids in developing messages	+ Aids in processing messages

TABLE 2-1. Characteristics of Computer Based Message Systems

Thus, from some humble origins, electronic mail was continually enhanced until it became almost a separate field, called either a computer-based message system

(CBMS) or a message handling system (MHS). While there are many different versions of CBMSs, they all must adhere to several basic concepts to be considered a CBMS. Each involves an originator and a recipient that are usually people or may also be processes, but they may not be specific terminal addresses. If the recipient system is not available when the message is sent, the originator's system holds the message until the recipient's system becomes available or a time-out occurs. Once the recipient CBMS receives the message, it stores that message until the recipient chooses to process it. The CBMS must include aids for the originator in developing the message and to assist the recipient in reading it [FIP]. Table 2-1 provides a summary of the characteristics of a CBMS.

2.3 The need for standards

The effectiveness of any CBMS depends primarily on how many people use it. This implies that mail systems must be able to interconnect to one another in order to maximize the use of each one of the individual systems. To accomplish this, the interface between any two local systems must be standardized. This does not imply that the two systems must process their messages in the same way. Indeed, their user interfaces, and the complexity of the options offered to the users may be entirely different, but when the message is ready to be sent,

all CBMSs must deliver it in the same format. Redell and White [RDL] addressed this problem of interconnection when they published a general architecture for CBMSs.

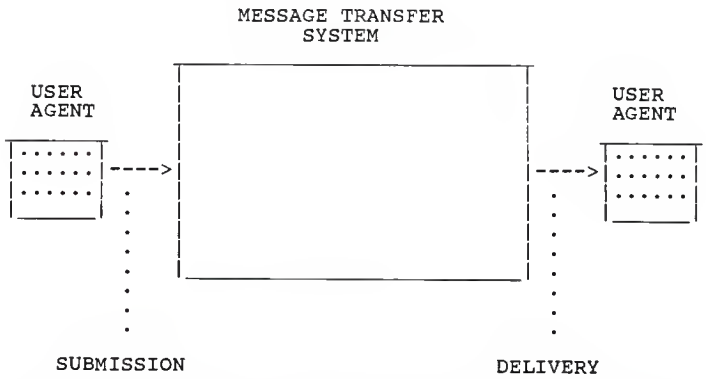


Figure 2-1. Components of a CBMS on a single machine

Figure 2-1 shows the different components of a CBMS according to the model developed by the International Federation for Information Processing (IFIP) [RDL]. A User Agent (UA) provides the interface to the user. It accesses an editor so the user can prepare a message and submits that message to the Message Transfer Service (MTS). The MTS acts as an "electronic post office" by transferring the message from the originating UA to the destination UA which accepts and stores the message. Here the UA also provides the interface so the user can read and process the message.

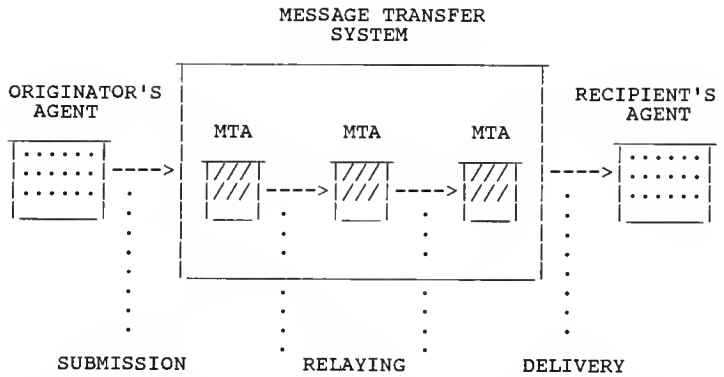


Figure 2-2. Components of a CBMS on multiple machines

Figure 2-1 is an example of a CBMS on a single machine. Figure 2-2 shows the added complexity of crossing over machine boundaries. The MTS must consist of several Message Transfer Agents (MTA), which cooperate to provide a store-and-forward path between any two UAs. A UA submits a message to a nearby MTA. This MTA relays the message to another MTA until the recipient's MTA is reached, whereupon the message is delivered to the recipient's UA.

Redell and White point out that for this system to work, each message must adhere to a standard format. Because the content of a message can vary so greatly from message to message, the CBMSs make use of a concept similar to that of a post office. The body of each message (the letter) is placed inside an

"envelope" which contains the recipient's identification and address. As with the post office, the MTA needs to look only at the "outside" of the envelope to see where the message should be sent. The envelope may change between the different MTAs depending on what information is needed to complete the delivery from that point. Because it is clearly separate from the envelope, the message body itself is left untouched. The body can be any type of digital information, intended for either humans or machines.

Despite the simplicity of the envelope architecture, there are many problems associated with interconnecting two or more networks. The first is the functional differences of the local MTAs. One of them, for example, might provide a confirmation of message delivery while another does not. The one that does would always expect to receive this confirmation back whenever it sends a message. But since this may never be received from some of the other MTAs, it could result in the first MTA re-sending the message. A standard that provides the functionality of the interface would eliminate these kinds of problems. The standard must include only those features that are essential for the functionality and be general enough to allow for flexibility as systems progress and become more sophisticated.

It was already mentioned that as long as the message envelope is readable by all the MTAs and UAs, that the message body could be of any type of data format. However, problems arise when, for example, numeric data is being passed from a 32 bit to a 16 bit UA. This raises the question of who should convert the messages and to what. One approach is to have all UAs convert all their messages to the "lowest common denominator" [RDL]. This way, all UAs, no matter how simple, would be able to interpret the messages. This ensures consistency, but keeps the more sophisticated UAs from taking full advantage of the CBMS, even when they communicate with one another.

A second approach suggested by Redell and White is called the "universal superset". Here, the standard format is so general that all the UAs can represent their data. This sacrifices consistency of service, but allows sophisticated UAs to use the electronic mail system to the fullest extent. If there is an intersystem directory that lists the capabilities of each UA, then the originator UA can translate the message before it sends it if the recipient UA does not recognize the format.

A problem with interconnecting CBMSs that may be more readily apparent to the user is that of naming

conventions. Different mail systems identify their users with a wide variety of naming patterns. It would be ideal if all mail systems used an international naming convention. However, since this is not likely in the near future, a more easily implemented solution must be found. One suggestion that is being used frequently now is to use a hierarchy of two-part names: the high order part contains the domain, and the low order part can be interpreted only within that domain [RDL]. Each MTA needs to recognize only the domain name to send the message on to the next MTA.

A problem that is related to naming conventions is that of distribution lists. Each MTA must be able to recognize the names within the distribution list of any users for which it is responsible. All the recipients should be handled uniformly and if there are nested distribution lists, then some mechanism must be developed to prevent recursive cycles from occurring.

Perhaps the problem with the most potential for dissatisfaction is that of security. Users must be confident that their messages go to the intended recipient and to no one else. They also must feel secure that no one can forge their name and send a message in their name. And conversely, recipients must be confident that the message they receive is actually

from the person marked as the originator. This is an area that will need to be addressed continually; as electronic mail becomes more sophisticated, the security measures will also become increasingly complex.

I. Cunningham [CUN] noted several additional factors that needed to be considered in developing the standards for a message-handling system.

- the standards had to support a range of messaging applications of which interpersonal messaging was the most important;
- the standards should not constrain future evolution;
- a variety of institutional domains (e.g. public and private) would be involved;
- regulatory constraints vary between countries;
- internetworking with previously existing message services (e.g. Telex and Teletex) were needed;
- different types of physical configurations would be used to implement the CBMS.

2.4 The development of standards

In the late 1970's and early 1980's, several different organizations attempted to address the problems mentioned above. A considerable amount of groundwork was laid by the International Federation for Information Processing (IFIPS). The International Telegraph and Telephone Consultative Committee (CCITT) used the IFIPS results to develop a standard known as X.400. CCITT's Study Group VII approved the recommendations in 1984. All of the conventions and models mentioned in this paper follow the X.400 Recommendations.

The National Bureau of Standards (NBS) also addressed the issue of interconnecting mail systems. In addition to following the IFIPS model, they published standards for the format of the messages. When these specifications were being developed, there were three major design perspectives that helped shape the format of the messages [FIP]. The first was viability; the developers of the standards used concepts that were already working. Thus the final product was something that could actually be implemented, rather than just a theoretical idealism. The second was compatibility; they used concepts from existing CBMSs. Many CBMSs already had functions and components similar to those required by the standards and therefore needed to make

only a few changes to meet the full specifications. The third was extensibility. The objective of the group here was to define a broad range of message content components, and then to make only an elementary subset of them actually required. This allows a simple CBMS to implement the message format specification while allowing for more sophistication in other and future CBMSs.

The overall objective of an electronic mail system is to send messages from an originator to a recipient. A message is simply one unit of communication that consists of a series of components called "fields". Fields can be described according to their meaning (semantics) or according to the format required for them in a message (syntax). The syntax of each field is best left to a fully detailed description of the standard, [FIP] however a basic definition of each field in the NBS specifications is given in Appendix A. The fields are listed by categories: required, basic, and optional. Required fields must appear in every message; basic fields must be recognized and processed by all CBMSs; and optional fields need not be supported by a CBMS but, if supported, must be processed according to the meanings defined by the message format specification.

In order not to limit the usefulness and applicability of the standards, they do not address:

- functions or services provided to a user
- storage or format of message contents in a CBMS
- message transfer system protocols
- message envelopes (headers used by the transfer system)
- how originators and recipients are identified.

2.5 Developments based on the standards

2.5.1 Connecting two CBMSs The objective in developing the X.400 Recommendations was to allow independent CBMSs to communicate. The ultimate test of this would be to develop two separate systems, ideally without any prior knowledge of the other. This is exactly what happened in 1982-1985. Kawaguchi et. al. [KAW] reported on the development of a CBMS in Japan and one in British Columbia, both starting in 1982. Both development teams were consistent with the CCITT X.400 Recommendations and accidentally learned of the other in the Spring of 1984. After several discussions on the specifications for interconnection, a test was performed from January to March of 1985 and messages were successfully sent and received. The success of this test demonstrates the usefulness of the standards.

2.5.2 Enhancing the UAs The X.400 Recommendations established the protocols of the message transfer service between two systems, without specifying the exact functionality of the User Agent. A great deal of effort has been put forth on enhancing both the originating and recipient user agents. One of the areas that has received much attention is that of distribution lists. Because it is easy to send electronic mail, people tend to send their messages to "anyone that might be interested". This results in people receiving messages about which they are only vaguely interested. One way to resolve this is to set up distribution lists that filter the list of potential recipients, searching for those that are specifically interested in the topic of the message. J. Palme [PL1] and T. Malone et. al. [MAL] have reported on research in this area.

2.5.3 Incorporating a database

2.5.3.1 Imail The work being done on the distribution lists is an example of an enhancement to the originator's UA. The work for this imail project focused on the recipient's UA. Both of them strive to achieve similar results, and that is to aid the user in filtering and organizing the messages so that those of importance can be readily recognized. Imail stores the messages a user receives in a database. The user can

then perform a set of predefined queries on the database. This set consists of queries that can be made about all or part of the fields that are either "required" or "basic" in the NBS specifications. (There is one exception to this, and that is the "reply-to"field. This field, however, is used for outgoing messages.) The queries that are permitted are:

- retrieve all messages by a particular author
- retrieve all messages with a particular keyword in the subject
- retrieve all messages with a particular keyword in the text
- retrieve all messages with a particular person in the "copy-to" list
- retrieve all messages received relative to a particular date and time
- combinations of the above

The messages are retrieved in the standard mail format so the user can process the messages in the manner allowed by the Unix mail command. Imail permits the user to specify a subset of messages to be viewed, and then does the filtering automatically.

Imail also provides for a great deal of additional flexibility for a user familiar with INGRES commands. The messages are stored in an INGRES database so the user may develop routines to manipulate them in ways not provided by imail.

2.5.3.2 Archiving service The concept of treating a mail message as a database record is also used by M. Tschichholz [TSC]. One of his objectives is to add an archiving service to the User Agent so it can relate messages to one another. This can be used on both the receiving and originating end.

Tschichholz refers to a document as an object that is being prepared for output. Messages are objects which have been transmitted and received and a message may be comprised of more than one document. An object (i.e. message or document) can be archived for long term storage into one or more "folders", which may be thought of as "in-baskets" and "out-baskets". Each user may access his own archives only, and may organize his folders himself. Objects that are no longer needed may be deleted by the user only.

A document contains the text of a message and the following header fields:

- "message id"

- "author"
- "title"
- "revision of"
- "reference to"
- "in reply to"

The "revision of" field shows the history of a document by maintaining a pointer to the original document. The "reference to" field contains a list of other documents and may be updated by the user. When the user answers a message, the identity of the message to which he is referring is indicated in the "in reply to" field of the header.

In addition to those fields, the user can assign to the objects:

- keyword (subject)
- textual remarks
- expiration date
- re-submission date (re-submitted objects will be entered in the "in-basket" folder again)
- explicit references to other objects

The archiving system assigns the object length and the time and date the object is stored. It allows an object to be archived into several folders and keeps track of all entries.

The user can ask to "leaf" through a folder and can view the objects in any of the following orders:

- chronological according to
  - production date (for documents)
  - reception time (for messages)
  - deposition time
- alphabetical by one of the header attributes.

Any of the sorting orders may be viewed forwards or backwards and the user may easily alternate between listing and displaying the objects. The user may also search in one or more folders for archived objects. The available search criteria are all the relevant header and envelope attributes (i.e., author or title) as well as information produced during the archiving of the objects (i.e., keyword, filing time). Searches for arbitrary character strings within the text are allowed as well.

Besides search functions, the User Agent provides functions for comparing objects. The basis for comparing messages is the list of documents found in each message. All equivalent messages can be identified as well as those that are subsets of another.

The system developed by Tschichholz is more complex than imail and thus it offers more options to its users. These options mainly involve the comparison of objects. But the archiving service and imail are similar in many ways. Both of them allow the user to easily switch from listing to displaying messages. Both allow for searching on header fields and arbitrary character strings in the message text. Imail does not make use of folders to organize the messages, but it does allow the user to set up a default ordering of messages. Once imail has been invoked, the user can dynamically select any subset view.

2.5.3.3 Application to the military The concept of using a database for the recipient's User Agent is being employed for an entirely different application than that mentioned above. T. Lunt [LUN] is using a database to maintain a secure mail system for the military. The architecture she is using is independent of any particular database or database architecture. Instead, there is a layer of "trusted" software that

translates requests into database queries and operations. Figure 2-3 shows the basic architecture.

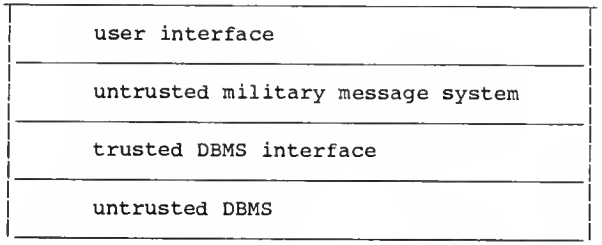


Figure 2-3. Architecture of a secure mail system

Her message system is similar to imail in that they both associate one message with a database record and have separate fields for the header information. However, to make the system secure, she must also add a classification, or military authorization, to each message. This is based on internal database files that map user identifiers to a classification. The trusted DBMS interface then uses the classification to limit the queries that may be made on the database.

2.5.3.4 Computer conferencing One of the most widespread uses of a database in conjunction with a CBMS is with computer conferencing systems. A "conference" commonly serves one of two purposes. The first is to provide a group mail service whereby a self-selected set of people can enter and read public

messages about a given topic. The other is to build a conference that will replace the need for the participants to physically meet together.

Unlike a CBMS, there is no "hard and fast" definition of the components and functions of a conferencing system. In general, they allow a user to "subscribe" to a particular conference(s), or topic(s). The user, then, may read all the messages posted to that conference by any other subscriber and may, in turn, add his own messages. He may even choose to "whisper" a message to another participant without being "overheard" by the other members [TWN]. The messages are stored in a large, usually centralized, database, and are not under the individual control of the user since they are considered to be part of the system resource.

Normally, when a user logs into a conference, he is notified of any new messages that have been posted since he last logged in. At any time, he may request to see a list of all the entries in the conference. In this way, new subscribers can be brought up to date quickly about the events that occurred before he participated. Entries within a conference are often structured into sets relating to particular subjects. The user interface generally allows actions to be

applied to sets of entries, as well as to individual entries (e.g. skip all entries in the current set) [KIL].

Because the messages are stored in a database, conferencing systems tend to exist only on large centralized facilities [KIL]. But it is the DBMS that keeps track of which messages are new to a user, which messages relate to one another and who the current subscribers are. It is the database that makes the conferencing systems possible.

Conferencing is just one of the many ways databases can be used to enhance electronic mail. Other applications strive to increase the functionality of the User Agents for individual users. These, and other advances, have expanded the concept of electronic mail beyond its original intent such that it is now a basic tool of the industry.

3. Implementation of imail

3.1 Introduction

The imail project focused on enhancing a recipient's User Agent for a local mail system. This chapter describes what that enhancement is, the environment in which it was developed, and how it was implemented.

3.2 Overview

The Unix "mail" command is a powerful electronic mail tool. The editing capabilities which the originator can use to build a message are quite impressive, and the processing that can be done on a message after it is received are equally varied. Yet mail currently does not allow a user to "preprocess" messages before viewing them. The focus of this project is to integrate a database management system (INGRES) with the functions of mail in order to allow the user greater flexibility in processing mail messages.

The set of routines that comprise the enhancement to mail is called "INGRES-mail" or simply "imail". Imail allows the user all the functionality of mail in addition to the ability to query the database for specific messages. The user can also dictate in advance the order in which the messages will be displayed by setting up priorities which are based upon

the author, a member of the "carbon copy" list, the date in the message, or a keyword in the subject heading or text. Those messages with the highest priority will be displayed first, followed in order by those of decreasing priority.

In addition to providing increased functionality for the mail command, imail establishes a platform for the user to write his own routines to manipulate his mail messages. The user is thus allowed direct access to the stored mail messages. This is discussed further in the section about the imail environment and in chapter four.

3.3 The user's view of imail

When a user wishes to read or process the messages in his mailbox, he may invoke imail rather than mail. Imail sorts all the messages according to an order the user has described, and then executes mail on those messages. The user may then use the standard mail commands to respond to, save, delete, etc. any of the messages. Every command allowed by mail may be executed within the imail setting. When the user "quits" the mail command, he is brought back into the imail environment.

Once in the imail environment, there are many functions

the user may select:

- search for messages from a particular author
- search for messages with a particular keyword in the subject
- search for messages with a particular keyword in the text
- search for messages with a particular person in the "copy-to" list
- search for messages relative to a particular date/time
- combinations of the above.

The user may select the output from each of the above functions to be in one of two formats. The first is a simple listing of the author, subject, and date of each of the matching messages. The second creates a subset of all messages that match and then invokes mail on that subset. As before, the user may perform any standard mail function and upon quitting is again in the imail environment.

There are two other functions that imail provides, the first of which is to reset the default order of the messages. The messages of all imail users are sorted

```
enter: A(ll), T(emp sort), R(eset sort order),
L(ist current sort order),
S(pecial search), K(ill me), or Q(uit for now)
> s
enter  A(nd), O(r), (, ),
or  one of the following and then the value:
F(rom), C(c), S(ubj), M(sgtext),
D(ate) (yy mm dd hh mm), Q(uit)
> subj paper
Enter (h) to see headers only, (m) to invoke mail,
(q) to quit request:   h

   beth          Tue Jun  7 13:45:12 1988   Re:  paper

   beth          Thu Jun  9 07:46:05 1988   Re:  paper

   vanburen      Mon Jun 13 09:27:04 1988   paper

3 messages found.


enter: A(ll), T(emp sort), R(eset sort order),
L(ist current sort order),
S(pecial search), K(ill me), or Q(uit for now)
> r
What do you want to sort on:  A(uth), S(ubj), C(c),
T(ext), or D(ate)? >
What do you want to be shown first?
Use (;) to quit > rich
next? > virg
next? > beth
next? > ;
sorting on 3 values

enter: A(ll), T(emp sort), R(eset sort order),
L(ist current sort order),
S(pecial search), K(ill me), or Q(uit for now)
> q
```

Figure 3-1. Examples using the imail functions

by the date of the message until the user specifically overrides this with a new ordering. This new order then remains in effect until the user requests to change it. Once this function is selected, the user is

prompted for the sort key. Date of the message, author, member of the carbon copy list, a keyword in the subject heading, or a keyword in the text are all valid sort keys. All of the options, except the date, require additional prompting for each of the keywords used for sorting. As an example, assume the user selected a sort by the authors "wood", "grebe", and "black". The next time imail is run, all the messages from "wood" are at the top of the list, followed by those from "grebe" and then those from "black". The messages from any other authors appear after those from "black". The secondary key used for sorting is the date of the message.

The last function of imail allows the user to request mail using a different sorting order but without changing the default order. The messages are sorted immediately and then mail is invoked with the new order. As before, any of the mail commands are permitted, and upon quitting mail, the user is still in imail. The sorting options for this function are the same as those of resetting the default order.

Figure 3-1 shows two examples of using imail. The first example is a compound request for all the messages that are from "rich" or those that are both from "beth" and have the word "paper" in the title.

The second example shows how to set the default sort order. Additional examples can be found in the Users' Manual in Appendix D.

3.4 Why use imail?

Imail is not a useful tool for those people who keep only a few messages at one time in their mailbox. It is, however, appropriate for those people who prefer to store their messages in their mailbox until the messages are no longer relevant. Imail allows them a method to keep track of all their messages and to easily select those that are of importance at the moment. Essentially, it allows them to use their mailbox as a miniature DBMS.

3.5 The imail environment

Imail was designed to run on the Unix operating system for a very practical reason. The project was developed at Kansas State University on a VAX* 11/780. This is the machine that most students and instructors commonly log into and hence the machine most frequently used for sending mail messages. To make imail accessible to many people, it was developed with Berkeley Unix 4.2, the operating system on the VAX.

---

* VAX is a trademark of Digital Equipment Corporation

One of the reasons imail was developed was to make the contents of a user's mailbox available to him in many different ways. This, coupled with the capabilities of imail listed in Section 3.3, required storing the mail messages outside the user's mailbox. It did not, however, necessarily mandate using a database management system. Another option available was to use a file processing system.

Imail could have been implemented with a file processing system by storing the user's mail messages in one or more files. The access to these files probably would have been faster than going through a DBMS. It also would have required less space. But these advantages fade when compared to the ease of using a DBMS for this and future work. It is important that users be able to easily access their stored messages outside of the imail environment. If a file processing system were used, each of these users would need to know the exact layout of the file, including field names and data types. If, in the future, any additional fields are added to the stored message, (for example, a field for the circulate-to list) then any routine that reads the files would need to be updated to reflect the new structure definition. Because this is not backward compatible, a great deal of coordination would have been needed among the users

when an update was made to the structure definition.

On the other hand, by using a DBMS, a new field that is added has no effect on previously written routines. They can continue to function exactly as they did before the database layout was updated. Furthermore, a DBMS hides the exact structure of the data from the user, who needs to know only the names of the fields and their type, but not their relative order. And, the user needs to access only those fields of interest to him, rather than the entire record.

INGRES was selected as the DBMS for imail for several reasons. The first, although important, is perhaps mundane: INGRES is supported on the VAX. But INGRES is a good choice also because it has a relatively simple user interface. Since there may be multiple users developing their own routines in the future, it is important that the data be easy to conceptualize and easy to access. Furthermore, INGRES allows even greater flexibility for the user in that it can be utilized as either a stand-alone system or may be accessed within an application program (this interface is known as Embedded Query Language, or EQUEL). For the imail project, EQUEL was accessed from programs written in the C programming language.

3.5.1 Cooperation between imail and Unix mail When people are faced with the prospect of using a new system, there is usually some degree of hesitation. Two questions frequently asked are: "Is it hard to learn?" and "Do I have to give up the old way entirely?" One of the advantages of using imail is that the answer to both these questions is "no."

Imail is easy to learn because it cooperates with Unix mail. There are only a few imail commands that must be learned and then when the user wishes to view his mail messages, he is put right back into the old comfortable mail environment. Thus even in the midst of imail, the user will have a sense of familiarity.

The user may elect to alternate his use of imail with the mail command. This is permitted with one note of caution. Once imail retrieves a message from the user's Unix mailbox, the only way to delete it is within the imail environment. Assume a user calls imail and is shown three messages. Then he quits imail, calls mail, and deletes one of the messages. The next time he calls imail, he will again see all three messages because the message had been deleted from the Unix mailbox only and not from the imail database.

3.6 The design of imail

3.6.1 Flow control Appendix B shows a high level flow control diagram of the imail process. Figure 3-2 shows how the imail and mail environments interact with one another.

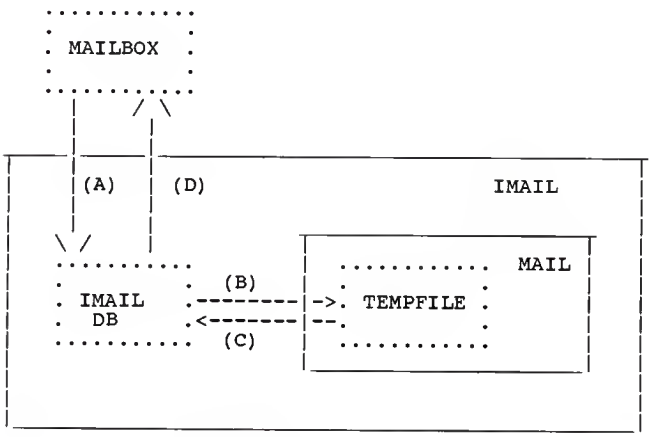


Figure 3-2. Relationship of imail and mail environments

Several things happen when a user invokes imail. First, imail fetches the default sorting order out of the database. Then it retrieves all the messages that were stored for that user in the database. If there is no entry in the database for this user it must be the first time he has invoked imail. An empty relation is created and the sorting order of new messages will be based upon the date of the message. Processing then

continues as it does for experienced users in that imail checks for new messages in the user's mailbox (A of Figure 3-2). At this point, the user's mailbox is emptied so that if any new messages arrive during the imail session, the user can be notified when he exits imail. Once imail has all the current messages, they are sorted according to the user's default order, put in a temporary file, and sent to the standard mail command (B of Figure 3-2). Any of the mail commands may be run, including deleting a message. When the user quits mail, the messages that were deleted from the temporary file are deleted from the imail database (C of Figure 3-2). Any of the imail commands may be called and with each of them, except for resetting or viewing the default order, the same steps are executed: the proper subset of messages is identified and stored in a temporary file, the temporary file is sent to the mail command, and the messages deleted from the file with mail are deleted in the imail database.

When the user quits imail, all the messages remaining in the database are sorted by date and copied back into the user's mailbox (D of Figure 3-2). Any new messages that may have arrived are preserved and the user is notified of the new mail.

3.6.2 Contents of the database The imail database consists of many different relations. There is one master relation that contains an entry for each imail user, his default sorting order, and the last time he used imail. There is also one relation per user that stores all the mail messages that have been retrieved from that user's mailbox. As the messages are retrieved from the mailbox, they are broken down into their component parts (subject, author, text, etc) and these individual fields make up the record in the imail database.

3.6.2.1 Some limitations of INGRES INGRES places constraints on the length of a field and the length of a record. This meant that most mail messages had to be broken up into several records and linked together. The effects of this are noted in the user's manual.

INGRES also does not handle variable length fields. To allow for this, truly variable length fields (e.g. cc-list) had to be treated in such a way that they could grow quite large. This was accomplished by allowing them to fill an entire record, if necessary.

INGRES does not recognize a small subset of special characters (e.g. "control L") and will not store them in the database. These characters must be "masked" before they are stored in the imail database.

3.7 Summary

Imail is a useful tool for people who use their mailbox to store messages. The pre-sorting function allows them to consistently give a high priority to messages from particular authors. Then regardless of how long their mail list grows, the messages from these authors always appear first. At any time, a user can re-sort the list or ask for a subset of messages. If these functions are too limiting, a user may access the messages in the imail database directly. Imail was designed to allow the user a great deal of flexibility in processing his mail messages.

4. Conclusions

4.1 Enhancements to imail

The functions of imail extend the capabilities of Unix mail. Those functions, while useful in and of themselves, are not the primary motivation for developing imail. Its greater usefulness lies in the basis of imail itself and that is the concept of storing mail messages in a database. With this accomplished, it is now possible to use the database to go beyond simply allowing a user to organize his mail messages according to the order best suited for him.

One possible enhancement to imail is to allow users to add comments to the mail messages in his mailbox. This would enable him to keep any notes about the message right there with the message so that both could be viewed at the same time. There are several ways this could be implemented. One is to keep these comments separate from the associated Unix mail message and view them through imail only. A second implementation technique is to actually append them to the mail message.

Another enhancement to imail is to allow the user to link one message with another. Every time one message is accessed, all the other messages linked to it could

automatically be available for viewing as well. In essence, this would permit the user to set up his own keyword for each message. This keyword could also be used as one of the options for fields upon which to base a sort. A keywords option could be developed into a simplified version of the archiving service by Tschichholz mentioned in chapter two. His system, though, was developed for both the receiving and the originating User Agents; imail was originally intended for the receiving UA only.

An enhancement that begins to cross the boundary into the originating UA is that of assisting the user in developing distribution lists. Imail could be modified to find messages about a given topic and to prepare a list of the authors of those messages and any members in the "copy-to" list. The user could then send messages about the chosen topic to the people in the list.

Distribution lists have received quite a bit of attention in the current literature. Many authors discuss the problems associated with the actual sending of the message to multiple recipients, [WOS] [PL1] while others address the problems of sending "junk mail" to too many recipients [PL2]. What has been suggested here is a very simple way to develop a

distribution list for personal use. A suggestion by D. Deutsch [DEU] is to allow the user to make modifications to the lists. This should be one of the requirements of imail were it to generate distribution lists.

The enhancements mentioned above may be implemented by modifying imail itself and extending its database capabilities. It is also possible, however, for users to directly access the imail database. A user would run imail to fetch messages from the Unix mailbox and properly load the database. But after that, the user may manipulate the imail database at will. This feature does jeopardize the integrity of the database and thus of imail, but it is a risk worth taking because it allows virtually unlimited possibilities for the use of the mail messages. It is assumed that a user interested enough in using the imail database will be careful enough not to compromise its integrity. It is also possible to recover from a "disaster" by requesting to be removed from the imail system entirely. This removes all references of the user who can then log back in again as a "new user". One other form of protection is that a user is allowed to access only his own messages in the imail database.

4.2 Non-textual messages

Computer based message services are advancing into the area of multi-media communications. Among these are both voice mail systems and video display systems. Both of these currently require separate systems that carry and interpret the appropriate type of traffic. An interface to one of these systems may some day be able to catalog the current messages. That is, it would interpret the originator, the date of the message, and possibly the subject and any of the other header fields and store them in some on-line database. The user then would be able to access a listing of all the messages in his mailbox and could peruse the headings in a textual fashion. This is generally faster than doing a sequential scan on the messages themselves.

4.3 Directions of development for electronic mail systems

Imail used a database management system in order to increase the functionality of a recipient's User Agent. If it is expanded, it can aid an originator in preparing outgoing mail. The direction of future computer based message systems is to build tools, such as a DBMS, to do as much bookkeeping work as possible for the user. This will free the user from those tasks and allow him to concentrate on the real purpose of a

CBMS, and that is to easily correspond with other users.

Bibliography

[CHR] Chirlian, Barbara S., "Simple dBase II". Dilithium Press, 1984,

[CIT] CCITT Study Group VII, "Data Communication Networks Message Handling Systems, Recommendations X.400-X.430". October, 1984.

[CUN] Cunningham, Ian, "Message-Handling Systems and Protocols". Proceedings of the IEEE, December, 1983, pp. 1425-1429.

[DEU] Deutsch, Debra P., "Implementing Distribution Lists in Computer-Based Message Systems". Computer-Based Message Services, Smith, H. T. (editor). Elsevier Science Publishers B.V. (North-Holland). IFIP, 1984.

[FIP] "Announcing the Standard for Message Format for Computer-Based Message Systems". Federal Information Processing Standards Publication, March 1, 1983.

[GIT] Gitman, Israel, "Voice Mail and Competing Services". Computer Message Systems - 85, Uhlig, R. P. (editor). Elsevier Science Publishers B.V. (North-Holland) IFIP, 1986.

[HAW] Hawryszkiewycz, I. T., "Database Analysis and Design". Science Research Associates, Inc., 1984.

[ING] "An Introduction to INGRES". Relational Technology Inc., 1983.

[JAB] Jaburek, W., Sebestyen, I., "Computerized Message Sending and Teleconferencing on Videotex Through Intelligent Decoders, Smart Cards, and Optical Cards". Computer-Based Message Services, Smith, H. T. (editor). Elsevier Science Publishers B.V. (North-Holland). IFIP, 1984.

[KAW] Kawaguchi, K., Sato, K., Sample, R., Demco, J., Hilpert, B., "Interconnecting Two X.400 Message Systems". Computer Message Systems - 85, Uhlig, R. P. (editor). Elsevier Science Publishers B.V. (North-Holland) IFIP, 1986.

[KIL] Kille, Steve, "Integration of Electronic Mail and Conferencing Systems". Computer-Based Message Services, Smith, H. T. (editor). Elsevier Science Publishers B.V. (North-Holland). IFIP,

1984.

[LUN] Lunt, Teresa F., "A model for Message System Security". Computer Message Systems - 85, Uhlig, R. P. (editor). Elsevier Science Publishers B.V. (North-Holland) IFIP, 1986.

[MAL] Malone, T. W., Grant, K. R., Turbak, F. A., Brobst, S. A., Cohen, M. D., "Intelligent Information-Sharing Systems". Communications of the ACM, May, 1987, pp 390-402.

[OHM] Ohmura, H., Kamiyama, Y., Kobayashi, H., "Development of a Multi-Media MHS Based on CCITT X.400 Recommendations". Computer Message Systems - 85, Uhlig, R. P. (editor). Elsevier Science Publishers B.V. (North-Holland) IFIP, 1986.

[PL1] Palme, Jacob, "Distribution Agents (Mailing Lists) in Message Handling Systems". Computer Message Systems - 85, Uhlig, R. P. (editor). Elsevier Science Publishers B.V. (North-Holland) IFIP, 1986.

[PL2] Palme, Jacob, "You Have 134 Unread Mail! Do You Want to Read Them Now?". Computer-Based Message Services, Smith, H. T. (editor). Elsevier Science Publishers B.V. (North-Holland). IFIP, 1984.

[RDL] Redell, David D., White, James E., "Interconnecting Electronic Mail Systems". Computer, September 1983, pp. 55-63.

[TWN] Townsend, Carl, "Electronic Mail and Beyond". Wadsworth Electronic Publishing Company, 1984.

[TSC] Tschichholz, Michael, "Message Handling System: Requirements to the User Agent". Computer Message Systems - 85, Uhlig, R. P. (editor). Elsevier Science Publishers B.V. (North-Holland) IFIP, 1986.

[WIL] Wilson, Paul, "Structure for Mailbox System Applications". Computer-Based Message Services, Smith, H. T. (editor). Elsevier Science Publishers B.V. (North-Holland). IFIP, 1984.

[WOS] Wosnitza, Lothar, "Group Communication in the MHS Context". Computer Message Systems - 85, Uhlig, R. P. (editor). Elsevier Science Publishers B.V. (North-Holland) IFIP, 1986.

Appendix A

Fields specified in the NBS specifications

Required fields must appear in a message:

From	Identifies originator(s) taking formal responsibility for this message
Posted-Date	Time the message passes through the posting slot into a message transfer system
To	Primary recipients for a message

* * *

Basic fields must be recognized and processed by all CBMS systems:

Cc	Secondary recipients of a message (a "carbon copies" list)
Reply-To	Identifies recipients for replies to the message
Subject	Whatever information the originator provided to indicate the nature of the message
Text	Primary content of the message

* * *

Optional fields need not be supported by a CBMS but, if supported, must be processed according to the meanings defined by the message format specification.

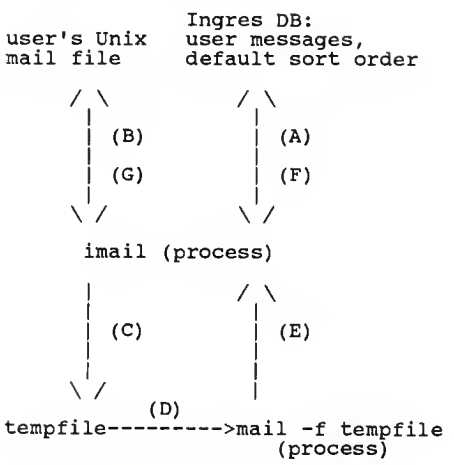
Attachments	Additional data accompanying a message; similar in intent to enclosures in a conventional mail system
Author	Identifies the individual(s) who wrote the primary contents of the message
Bcc	Identifies additional recipients of a message (a "blind carbon copies" list)
Circulate-Next	Identifies all recipients in a circulation list who have not yet received the message

Circulate-To	Identifies all recipients of a circulated message
Comments	Permits adding comments onto the message without disturbing the original contents of the message
Date	Date that the message's originator wishes to associate with a message
End-Date	Date on which a message loses effect
In-Reply-To	Designates previous correspondence to which this message is a reply
Keywords	Keywords or phrases for use in retrieving a message
Message-Class	Purpose of a message; i.e. it might contain values indicating that the message is a memorandum or a database entry
Message-ID	Unique identifier for a message; intended for machine generation and processing
Obsoletes	Identifies one or more messages that this one replaces
Originator-Serial-Number	One or more serial numbers assigned by the message's originator
Precedence	The precedence at which the message was posted
Received-Date	Time the message left the delivery system and entered the recipient's message processing domain
Received-From	A record of a message's path through a message transfer system
References	Identifies other correspondence to which this message refers
Reissue-Type	Differentiates between messages being assigned or redistributed
Sender	Identifies the agent who sent the message
Start-Date	Date on which a message takes effect

Warning-Date	Warning of an impending end-date or other event

Appendix B

Flow Control of Imail



- The user runs imail.
- (A) The user's default sort order and old messages are read from the DB.
- (B) New messages are read from the user's Unix mail file.
- (C) The messages are sorted in default order and written to the tempfile.
- (D) Mail -f is run on the tempfile. All Unix mail commands are allowed, including deleting a message.
- (E) When the user exits or quits mail, he is back in the imail environment. Messages that were deleted from within mail are deleted from the imail database. Any of the imail functions may be performed. (C), (D), and (E) may be repeated using different selection criteria.

- (F) If the default order is changed, the new parameters are written to the imail database.
- (G) When the user exits imail, the remaining messages are sorted by date and stored back in Unix mail file.

Appendix C

Application Code

Table of Contents

```
#include <stdio.h>
#include <signal.h>
#include <sys/time.h>
#include "imail.h"

/*****************************************************************
**********
        main.q

        This is the main driver for imail.  Routines found in other
        .c's
        start with a Capital letter.  Global variables do too.

*****************************************************************
*********/

FILE    *Tfp;   /* file pointer of temp mail file */
FILE    *bugfp;
char    nameholder[20];
##char  *Usrname;
##char  Imrel[64];       /* ingres msg file */
char    Mailfile[64];   /* usr's regular mail file */
char    Lmailfile[64];  /* file linked to usr's regular mail file */
char    Callmail[50];   /* sets up system call to mail */
char    Tempfile[32];   /* hold msgs usr is currently viewing */
char    Rmtemp[50];     /* sets up system call to remove Tempfile */
int     Stretch;        /* drop right into stretch functions */
int     Tilda;          /* translate ~ to ~~ and control chars to ~x
*/
##int   Sort_type, Snum_vals;           /* sort on auth, cc...; num
keys */
##char  Sort_vals[MAXKEYS][KEYLEN];     /* holds order of appearance
*/
int     nm;             /* new message flag */
int Numdk;

        /* this struct holds the key to all the msgs the user is
        currently
                viewing.  At the end of the viewing session (the end
                of
                mail) this list is compared with the messages still
                in the
                temp file.  Any msgs that have been deleted from the
                tempfile are deleted from the ingres relation.  Thus
                at any time (except during "mail") the ingres db
                holds
                only those messages that have not been deleted.
        */
##struct delkeep
##{
##      char iauth[AUTHLEN+1];
##      long idate;
##}Dk[200];
```

```
main(argc, argv)
int argc;
char *argv[];
{
        int     cmd, hdr_or_txt;
        char    qual[QUAL_LEN];
        extern int errproc();
        register int i;
        char temp[200];
        int getfullmsg; /* for cc and msgtext retrievals, rtrv all
        msg parts*/
        int yy;

        signal(SIGINT, SIG_IGN);
        umask(077);

        Usrname = (char *)getlogin();
#ifdef DEBUGNAME /* prompt for input name  (for debugging) */
        printf("enter user name:  ");
        fflush(stdout);
        scanf("%s", nameholder);
        Usrname = nameholder;
#endif
        /* set up the names of all the files needed */
        sprintf(Imrel, "im%.10s", Usrname);
        sprintf(Mailfile, "%s/%s", MAILDIR, Usrname);
        sprintf(Tempfile, "/usr/tmp/tm%s", Usrname);
        sprintf(Lmailfile, "/usr/tmp/lm%s", Usrname);
        sprintf(Callmail, "mail -f %s", Tempfile);
        sprintf(Rmtemp, "rm  %s", Tempfile);

#ifdef DEBUG
        if ((bugfp = fopen("debug", "w")) == NULL)
                printf("cant open debug\n");
#endif

#ifdef DEBUG5
        fprintf(bugfp, "Mailfile %s\n", Mailfile);
        fprintf(bugfp, "Imrel %s\n", Imrel);
        fprintf(bugfp, "Callmail %s\n", Callmail);
        fprintf(bugfp, "Tempfile %s\n", Tempfile);
        fflush(bugfp);
#endif

        /* Catch the options to imail */
        Stretch = 0;
        Tilda = 0;
        for (i=1; i<argc; i++)
        {
                if (strncmp(argv[i], "-s",2) == 0)
                        Stretch = 1;
                if (strncmp(argv[i], "-c",2) == 0)
                        Tilda = 1;
```

```
        }

#ifdef DEBUG5
        fprintf(bugfp, "Stretch = %d, Tilda = %d", Stretch, Tilda);
#endif
        /* verify the call was correct */
        if ((argc == 2 && (Stretch + Tilda != 1))  ||
            (argc == 3 && (Stretch + Tilda != 2)))
        {
                printf("Valid options are : -s   to immediately
                access ");
                printf("stretch functions\n");
                printf("                    -c");
                printf("   to translate control characters\n");
                fflush(stdout);
                exit();
        }
##      ingres imaildb

        New_msgs();     /* read new msgs from USRMAIL (if any), put
        in idb */

        /* invoking "imail -s" means the user wants to use a Stretch
        function
           before looking at the messages.  (or he wants to Skip the
           mail part)
        */
        if (!Stretch)
        {
                sort_to_temp(qual);/* use dflt order, put msgs in
                temp file */
                system(Callmail);       /* run mail */
                Temp_to_idb(Tfp);       /* put remaining msgs back
                in idb */
                system(Rmtemp);         /* don't need Tempfile
                anymore */
        }
        /* loop until the user wants to quit */
        while (Get_cmd(&cmd, &hdr_or_txt, qual, &getfullmsg) ==
        GO_ON)
        {
                stretch(cmd, hdr_or_txt, qual, getfullmsg);
        }

        /* write all idb msgs into USRMAIL */
        if (Idb_to_mail())
                printf("New mail arrived\n");

        /* see if user drops out forever */
        if (cmd == KILL)
        {
##              destroy Imrel
##              range of ilog is logrel
##              delete ilog where (ilog.usrname = Usrname)
                printf("You have been deleted from the imail
                database\n");
```

```
        }

        /* remove the link file */
        /* if you're debugging, you may want to look here */
        sprintf(temp, "rm %s", Lmailfile);
        system(temp);

#ifdef DEBUG
        fclose(bugfp);
#endif

##      exit
}

/*****************************************************************
**********

        all_hdrs

        Show the headers to all the messages.

*****************************************************************
*********/
all_hdrs()
{
##      char    iauth[AUTHLEN+1], isubj[SUBJLEN+1];
##      long    idate;
##      int     iseqnum;
        char    *datestr;

##      range of idb is Imrel
##      retrieve(
##              iauth=idb.auth, isubj=idb.subj,
##              idate=idb.date, iseqnum=idb.seqnum)
##              where idb.seqnum = 0
##      {
                addnull(iauth, AUTHLEN+1);
                addnull(isubj, SUBJLEN+1);
                datestr = (char *)ctime(&idate);

                printf("\n   %s", iauth);
                if (strlen(iauth) < 5)
                        printf("        ");     /* line things up */
                printf("        %.24s", datestr);
                printf("   %s", isubj);
                fflush(stdout);
##      }
        fflush(stdout);
}
```

```
/*****************************************************************
**********

        app_tfile

        The qualifier was set up by the calling routine.  This does
        the
        retrieval and appends any messages it finds to the tempfile.

*****************************************************************
*********/
app_tfile(qual)
##char *qual;   /* qualifier for retrieve stmt */
{
##      int idbseqnum, idbtuplen;
##      char idbauth[AUTHLEN+1];
##      long idbdate;
##      char idbtext[MAXPARTS][TEXTLEN+1], idbtup_ty[2];
        int i, tpart, tpos, needit;

#ifdef DEBUG2
        fprintf(bugfp, "app_tfile   ");
        fprintf(bugfp, "qual = %s\n", qual);
        fflush(bugfp);
#endif
##      range of idb is Imrel
##      retrieve (
##              idbauth=idb.auth, idbdate=idb.date,
##              idbtuplen=idb.tuplen, idbtup_ty=idb.tup_ty,
##              idbtext[0]=idb.text0, idbtext[1]=idb.text1,
##              idbtext[2]=idb.text2,
##              idbtext[3]=idb.text3, idbtext[4]=idb.text4,
##              idbseqnum=idb.seqnum)
##              where qual
##      {
#ifdef DEBUG4
                fprintf(bugfp, "app_tfile: retrieve\n");
                for (i=0; i<MAXPARTS; i++)
                {
                        fprintf(bugfp,"\nkk%dkk:  %s", i,
                        idbtext[i]);
                }
#endif
                /* set up del-keep struct so can later del msgs */
                if (idbseqnum == 0)     /* only 1 entry per mail msg
                */
                {
                    /* need to get rid of the blanks that ingres put
                       in and null terminate the string.
                    */
                    addnull(idbauth, AUTHLEN+1);
                    needit = 1;
                    /* loop thru all msgs already found */
```

```
                    for (i=0; i<Numdk; i++)
                    {
                        if (Dk[i].idate == idbdate &&
                            (strcmp(Dk[i].iauth, idbauth) == 0))
                        {       /* it's already there */
                            needit = 0;
                            break;
                        }
                    }
                    if (needit) /* add this one to the list */
                    {
                        strcpy(Dk[Numdk].iauth, idbauth);
                        Dk[Numdk].idate = idbdate;
#ifdef DEBUG8
                        fprintf(bugfp,"Dk[%d] %s %D    ",
                                Numdk, Dk[Numdk].iauth,
                                Dk[Numdk].idate);
                        if (Numdk%2)  fprintf(bugfp, "\n");
                        fflush(bugfp);
#endif
                        Numdk++;
                    }
                }

                if (needit)
                {
                        tpos = idbtuplen % TEXTLEN;
                        tpart = (idbtuplen - tpos)/TEXTLEN;
#ifdef DEBUG7
                        fprintf(bugfp, "app_tfile:idbtuplen = %d
                        ",idbtuplen);
                        fprintf(bugfp, "tpart = %d, tpos = %d",
                        tpart, tpos);
#endif
                        for (i=0; i<tpart; i++)
                        {
                                idbtext[i][TEXTLEN] = '\0';
                                fprintf(Tfp,"%s", idbtext[i]);
                        }
                        if (tpos)       /* else nothing to write */
                        {
                                idbtext[tpart][tpos] = '\0';
                                fprintf(Tfp,"%s", idbtext[tpart]);
                        }
                }
##      }

}
```

```
/*****************************************************************
**********

        getfmsg

        If the user does a search on something other than auth or
        subject, then it's difficult to get all the parts of the
        msg.
        This routine makes a list of the key (auth, date) to any
        msg that matches the request and then goes thru that list
        to get all the records that make up that msg.

        In other words, if record 2 of the full msg matches the
        search
        string, you still need to retrieve records 1, 3, 4, ...

*****************************************************************
*********/
getfmsg(quall)
## char *quall;
{
##      char dbauth[AUTHLEN];
##      long dbdate;
        int gfnum, i;
        struct getfull
        {
                long gfdate;
                char gfauth[AUTHLEN];
                char gfnull;
        }gf[100];

        gfnum = 0;
##      range of idb is Imrel
##      retrieve (
##              dbauth=idb.auth, dbdate = idb.date)
##              where quall
##      {
                strncpy(gf[gfnum].gfauth, dbauth, AUTHLEN);
                gf[gfnum].gfdate = dbdate;
                gf[gfnum].gfnull = '\0';
                gfnum++;
##      }

        /* now you've got all the matching msgs, sort them */
        qsort(&gf[0].gfdate, gfnum, sizeof(gf[0]), strcmp);
        /* check for duplicates and append the originals to the
        tempfile */
        for (i=0; i<gfnum; i++)
        {
                /* set up qual for app_tfile */
                sprintf(quall, " idb.auth=\"%s\" and idb.date = %D",
                        gf[i].gfauth, gf[i].gfdate);
                if (i>0)
```

```
                {
                        /* dont do dups (remember, they're sorted */
                        if (strcmp(&gf[i].gfdate, &gf[i-1].gfdate)
                        !=0)
                                app_tfile(qual1);
                }
                else    /* append to the first one */
                        app_tfile(qual1);
        }
}

/*****************************************************************
**********

        re_set

        The user wants to reset the default sort order.  Prompt for
        the
        order and change the sort keys in the log relation.

*****************************************************************
*********/
re_set()
{
        register i;

        if (Prompt_order())             /* loads the Sort_vals */
                return;
##      range of ilog is logrel
##      replace ilog (sorttype = Sort_type, num_svals = Snum_vals,
##                      sval0 = Sort_vals[0], sval1 = Sort_vals[1],
##                      sval2 = Sort_vals[2], sval3 = Sort_vals[3],
##                      sval4 = Sort_vals[4])
##              where ilog.usrname = Usrname

}


/*****************************************************************
**********

        re_sort

        User requests a sort on a different order, but do not change
        the
        default order.

*****************************************************************
*********/
re_sort(qual)
char *qual;
{
        if (Prompt_order())             /* prompt for the order */
                return;                 /* invalid user entry */
        sort_to_temp(qual);             /* sort msgs into temp file
        */
```

```
        system(Callmail);               /* call the mail routine */
        Temp_to_idb(Tfp);               /* put remaining msgs back
        into idb */
        system(Rmtemp);                 /* don't need Tempfile
        anymore */
}

/*****************************************************************
**********

        some_to_temp

        This is called when the stretch function has set up the
        qualifier
        for a subset of messages.  First retrieve the keys so that
        duplicates
        can be eliminated.

*****************************************************************
*********/
some_to_temp(hdr_or_txt, qual, getfullmsg)
int hdr_or_txt;
char *qual;
int getfullmsg;
{
        register int i, j, num_msgs;
        char    *datestr;
##      struct msghdr
##      {
##              char auth[AUTHLEN];
##              long date;
##              char subj[SUBJLEN];
##      }mh[100];

##      int nummh;      /* number of message headers */

        if (hdr_or_txt == HDR_ONLY)     /* the user requested
        headers only */
        {
                printf("\n");
                num_msgs = 0;
                nummh = 0;
##              range of idb is Imrel
##              retrieve(
##                      mh[nummh].auth=idb.auth,
mh[nummh].subj=idb.subj,
##                      mh[nummh].date=idb.date)
##                      where qual
##              {
                        addnull(mh[nummh].auth, AUTHLEN+1);
                        nummh++;
##              }
                for (i=0; i< nummh; i++)
                {
```

```
                        for (j=0; j<i; j++)
                        {
                                /* already have it? */
                                if (strncmp(mh[j].auth,mh[i].auth,
                                AUTHLEN)
                                     == 0 && mh[j].date ==
                                     mh[i].date)
                                        break;
                        }
                        if (j==i)       /* dont have it yet */
                        {
                                datestr = (char
                                *)ctime(&mh[i].date);

                                printf("\n   %s", mh[i].auth);
                                if (strlen(mh[i].auth) < 5)
                                        printf("        ");
                                printf("        %.24s", datestr);
                                mh[i].subj[SUBJLEN] = '\0';
                                printf("   %s", mh[i].subj);
                                fflush(stdout);
                                num_msgs++;
                        }

                }
                printf("\n%d messages found.\n\n", num_msgs);
                fflush(stdout);
        }
        else    /* user wants to run mail on matching messages */
        {
                if ((Tfp = fopen(Tempfile, "w")) == NULL)/* creat
                temp file */
                {
                        printf("WARNING: can't open Tempfile\n");
                        fflush(stdout);
                }

                if (getfullmsg)  /* special retrieve to get all
                parts */
                        getfmsg(qual);
                else
                        app_tfile(qual);  /* put the msgs in the
                        mail file */
                fclose(Tfp);
        }
}
```

```
/*****************************************************************
**********

        sort_to_temp

        This version of ingres cannot do sorts.  And even if it
        could, it
        wouldn't help a whole lot.

        Retrieve all the msgs that match the user's highest priority
        sort
        key, then all the next, then the next, etc.

*****************************************************************
*********/
sort_to_temp(qual)
char *qual;
{
        char qual_fnl[QUAL_LEN];        /* qual for final pass */
        int charpos;
        register int i;

#ifdef DEBUG4
        fprintf(bugfp, "sort_to_temp:   \n");
#endif
        if ((Tfp = fopen(Tempfile, "w")) == NULL)
                printf("WARNING: cant open Tempfile\n");

##      range of idb is Imrel

        if (Sort_type ==  DATE) /* need just one call to app_tfile*/
        {
                /* They'll be sorted by date anyway, don't need to
                   do anything; make qual meaningless
                */
                sprintf(qual, " idb.seqnum >= 0");
                app_tfile(qual);
                fclose(Tfp);
                return;
        }

        charpos = 0;
        for (i=0; i<Snum_vals; i++)
        {
                if ((Sort_type == CC || Sort_type==TEXT))
                {
                        /* put in a useless qualifier; will have to
                        eliminate
                        dups after retrieval for these
                        */
                        if (i==0)
                                strcpy(qual_fnl, " idb.seqnum >=
                                0");
```

```
                }
                else if (i > 0)
                {
                        /* the final pass here can be qualified so
                        as
                           not to include any of the msgs that
                           have already been retrieved
                        */
                        strcpy(&qual_fnl[charpos], " and ");
                        charpos +=5;
                }

                switch(Sort_type)
                {
                    case AUTHOR:
                        sprintf(qual, " idb.auth = \"*%s*\"",
                        Sort_vals[i]);
                        /* prepare string for final pass */
                        sprintf(&qual_fnl[charpos],
                                " idb.auth != \"*%s*\"",
                                Sort_vals[i]);
                        charpos = strlen(qual_fnl);
                        break;
                    case CC:
                        sprintf(qual, " (idb.tup_ty = \"cc\" and
                        (idb.text0 = \"*%s*\" or idb.text1 =
                        \"*%s*\" or idb.text2 = \"*%s*\" or
                        idb.text3 = \"*%s*\" or idb.text4 =
                        \"*%s*\"))",
                                Sort_vals[i], Sort_vals[i],
                                Sort_vals[i], Sort_vals[i],
                                Sort_vals[i]);
                        /* no qual_fnl needed */
                        break;
                    case SUBJECT:
                        sprintf(qual, " idb.subj = \"*%s*\"",
                        Sort_vals[i]);
                        sprintf(&qual_fnl[charpos]," idb.subj !=
                        \"*%s*\"",
                                Sort_vals[i]);
                        charpos = strlen(qual_fnl);
                        break;
                    case TEXT:
                        sprintf(qual, "(idb.tup_ty = \"tx\" and
                        (idb.text0 = \"*%s*\" or idb.text1 =
                        \"*%s*\" or idb.text2 = \"*%s*\" or
                        idb.text3 = \"*%s*\" or idb.text4 =
                        \"*%s*\"))",
                                Sort_vals[i], Sort_vals[i],
                                Sort_vals[i],
                                Sort_vals[i], Sort_vals[i]);
                        /* no qual_fnl needed */
                        break;
                }
#ifdef DEBUG2
```

```
                fprintf(bugfp, "sort_to_temp: qual: %s\n", qual);
                fprintf(bugfp, "sort_to_temp: qual_fnl: %s\n",
                qual_fnl);
                fflush(bugfp);
#endif
                if (Sort_type == CC || Sort_type == TEXT)
                        getfmsg(qual);
                else
                        /* do the retrv and build the temp mail file
                        */
                        app_tfile(qual);
        }       /* end loop thru each sort key */

        if (Snum_vals == 0)     /* make the qualifier useless, but
        fill it in*/
                strcpy(qual_fnl, " idb.seqnum >= 0");
        app_tfile(qual_fnl);
        fclose(Tfp);
}

/*****************************************************************
**********

        stretch

        The stretch commands stretch the limits of normal mail.

*****************************************************************
*********/
stretch(cmd, hdr_or_txt, qual, getfullmsg)
int cmd, hdr_or_txt;
char *qual;
int getfullmsg;
{
        Numdk = 0;      /* init del-keep struct */

        switch(cmd)
        {
                case RE_SORT:           /* user wants a different
                sort */
                        re_sort(qual);
                        break;
                case RE_SET:            /* reset the default sort
                order */
                        re_set();
                        break;
                case ALL:               /* retrieve all msgs */
                        sort_to_temp(qual);/* dflt order, put msgs
                        in temp */
                        system(Callmail);
                        Temp_to_idb(Tfp);
                        system(Rmtemp); /* can remove it now */
                        break;
                case ALL_HDRS:
                        all_hdrs();
```

```
                break;
        case INVALID:           /* you figure this one out
        */
                break;
        default:        /* all others require a subset view
        */
                some_to_temp(hdr_or_txt, qual, getfullmsg);
                /* some_to_temp took care of HDR_ONLY case
                */
                if (hdr_or_txt == RUN_MAIL)
                {
                        system(Callmail);
                        Temp_to_idb(Tfp);
                        system(Rmtemp); /* can remove it now
                        */
                }
                break;
        }
}
```

```
#include <stdio.h>
#include "imail.h"

extern char *next();
extern FILE *bugfp;

/*   The routines found in usrman.c are those that interface with
the user
*/

/*********************************************************************
*********

        Get_cmd

*********************************************************************
********/

Get_cmd(cmd, hdr_or_txt, qual, getfullmsg)
int     *cmd, *hdr_or_txt;
char *qual;
int *getfullmsg;
{
        char temp[80], request[200];
        char date[20], less_great[3];
        char linebuf[81], *lptr;
        register int charpos, i;
        char year[3], month[3], day[3], hour[3], minute[3];
        int yy, mm, dd, hh, min;
        long dateval;
        extern char Sort_vals[][KEYLEN];
        extern int Snum_vals;

        charpos = 0;

        printf("\nenter: A(ll), ");
        printf("T(emp sort), R(eset sort order), ");
        printf("L(ist current sort order), \n");
        printf("S(pecial search), K(ill me), or Q(uit for now)\n>
        ");
        fflush(stdout);

        scanf("%s",temp);

        *getfullmsg = 0;

        switch (temp[0])
        {
                case 'q':
                case 'Q':
                        return (QUIT);
                case 'a':
                case 'A':
                        *cmd = ALL;
```

```
                        *hdr_or_txt = RUN_MAIL;
                        return(GO_ON);
                case 'h':
                case 'H':
                        *cmd = ALL_HDRS;
                        *hdr_or_txt = HDR_ONLY;
                        return(GO_ON);
                case 't':
                case 'T':
                        *cmd = RE_SORT;
                        *hdr_or_txt = RUN_MAIL;
                        return(GO_ON);
                case 'r':
                case 'R':
                        *cmd = RE_SET;
                        return(GO_ON);
                case 's':
                case 'S':
                        *cmd = STRETCH;
                        break;
                case 'l':
                case 'L':
                        printf("sorting order is:\n");
                        for (i=0; i<Snum_vals; i++)
                                printf("  %s", Sort_vals[i]);
                        printf("\n");
                        *cmd = INVALID;
                        return(GO_ON);
                        break;
                case 'k':
                case 'K':
                        *cmd = KILL;
                        return(QUIT);
                        break;
                default:
                        printf("Invalid command, try again.\n");
                        fflush(stdout);
                        *cmd = INVALID;
                        /* flush out anything else on this line */
                        gets(linebuf);
                        return(GO_ON);
                        break;

        }       /* end switch */

        getc(stdin);    /* get rid of the newline */
        strcpy(request, "");
        printf("enter  A(nd), O(r), (, ), or  ");
        printf("one of the following and then the value:\n");
        printf("F(rom), C(c), S(ubj), ");
        printf("M(sgtext), D(ate) (<> yy mm dd hh mm), Q(uit)\n> ");
        fflush(stdout);
        gets(linebuf);
        lptr = linebuf;
```

```
        while (lptr != NOSTR)
        {
            lptr = next(lptr, temp);
            switch (temp[0])
            {
                case 'a':
                case 'A':
                        sprintf(&qual[charpos], " and ");
                        charpos +=5;
                        strcat(request, temp);
                        strcat(request, " ");
                        break;
                case 'o':
                case 'O':
                        sprintf(&qual[charpos], " or ");
                        charpos +=4;
                        strcat(request, temp);
                        strcat(request, " ");
                        break;
                case '(':
                case ')':
                        sprintf(&qual[charpos], " %s ", temp);
                        charpos +=3;
                        strcat(request, temp);
                        strcat(request, " ");
                        break;
                case 'f':
                case 'F':
                        strcat(request, temp);
                        strcat(request, " ");
                        lptr = next(lptr, temp);
                        sprintf(&qual[charpos], " idb.auth =
                        \"*%s*\"", temp);
                        charpos = strlen(qual);
                        strcat(request, temp);
                        strcat(request, " ");
                        break;
                case 'c':
                case 'C':
                        strcat(request, temp);
                        strcat(request, " ");
                        lptr = next(lptr, temp);
                        sprintf(&qual[charpos], " ((idb.text0 =
                        \"*%s*\" or idb.text1 = \"*%s*\" or
                        idb.text2 = \"*%s*\" or idb.text3 = \"*%s*\"
                        or idb.text4 = \"*%s*\") and idb.tup_ty =
                        \"cc\")", temp, temp, temp, temp, temp);
                        charpos = strlen(qual);
                        *getfullmsg = 1;
                        strcat(request, temp);
                        strcat(request, " ");
                        break;
                case 's':
                case 'S':
                        strcat(request, temp);
```

```
                strcat(request, " ");
                lptr = next(lptr, temp);
                sprintf(&qual[charpos], " idb.subj =
                \"*%s*\"", temp);
                charpos = strlen(qual);
                strcat(request, temp);
                strcat(request, " ");
                break;
        case 'd':
        case 'D':
                strcat(request, temp);
                strcat(request, " ");

                lptr = next(lptr, less_great);
                lptr = next(lptr, year);
                lptr = next(lptr, month);
                lptr = next(lptr, day);
                lptr = next(lptr, hour);
                lptr = next(lptr, minute);
                sprintf(date, "%s %s %s %s %s",year, month,
                day,
                        hour, minute);
                yy = atoi(year);
                mm = atoi(month);
                dd = atoi(day);
                hh = atoi(hour);
                min = atoi(minute);
                strcat(request, less_great);
                strcat(request, " ");
                strcat(request, date);
                strcat(request, " ");
                dateval = (long)cnvtime(yy, mm, dd, hh,
                        min, 0);
                sprintf(&qual[charpos], " idb.date  %s= %d",
                less_great,
                        dateval);
                charpos = strlen(qual);
                break;
        case 'm':
        case 'M':
                strcat(request, temp);
                strcat(request, " ");
                lptr = next(lptr, temp);
                sprintf(&qual[charpos], " ((idb.text0 =
                \"*%s*\" or idb.text1 = \"*%s*\" or
                idb.text2 = \"*%s*\" or idb.text3 = \"*%s*\"
                or idb.text4 = \"*%s*\") and idb.tup_ty =
                \"tx\")", temp, temp, temp, temp, temp);
                charpos = strlen(qual);
                *getfullmsg = 1;
                strcat(request, temp);
                strcat(request, " ");
                break;
        case 'q':
        case 'Q':
```

```
                        *cmd = INVALID;
                        return(GO_ON);
                        break;
                default:
                        printf("Invalid command, try again\n");
                        break;
            }   /* end switch */
#ifdef DEBUG4
            printf("Building request: %s\n", request);
            fprintf(bugfp, "Get_cmd: qual = %s\n", qual);
            fflush(bugfp);
            fflush(stdout);
#endif
            /* ingres puts a limit on how long the 'where' clause
            can be */
            if (charpos>250)
            {
                printf("Sorry, the request is too long, try
                again\n");
                fflush(stdout);
                *cmd = INVALID;
                return(GO_ON);
            }
        }       /* end while loop */

        printf("Enter (h) to see headers only,");
        printf(" (m) to invoke mail,");
        printf(" (q) to quit request:   ");
        fflush(stdout);
        scanf("%s",temp);
        if (temp[0] == 'h')
                *hdr_or_txt = HDR_ONLY;
        if (temp[0] == 'm')
                *hdr_or_txt = RUN_MAIL;
        if (temp[0] == 'q')
                *cmd = INVALID;
        return(GO_ON);
}
/*****************************************************************
*********

        next

        Collect a liberal (space, tab delimited) word into the word
        buffer
        passed.  Also, return a pointer to the next word following
        that,
        or NOSTR if none follow.

*****************************************************************
********/
char *
next(wp, wbuf)
char wp[], wbuf[];   /* ptr to input str, ptr to word you are
looking for */
```

```
{
        register char *thisline, *newword;

        if ((thisline = wp) == NOSTR) {
                copy("", wbuf);
                return(NOSTR);
        }
        newword = wbuf;
        if (any(*thisline, "()<>"))
        {
                *newword++ = *thisline++;
        }
        else
        {
            /* it's legal to say (from robin) rather than ( from
            robin ) */
            /* copy until get special char */
            while (!any(*thisline, " \t()<>") && *thisline != '\0')
            {
                if (*thisline == '"')
                {
                        *newword++ = *thisline++;
                        while (*thisline != '\0' && *thisline !=
                        '"')
                                *newword++ = *thisline++;
                        if (*thisline == '"')
                                *newword++ = *thisline++;
                } else
                        *newword++ = *thisline++;
            }
        }

        *newword = '\0';
        /* fill in til hit end char */
        while (any(*thisline, " \t"))
                thisline++;
        if (*thisline == '\0')
                return(NOSTR);
        return(thisline);
}

/*********************************************************************
*********

        Prompt_order

        Prompt the user for the sorting order.

*********************************************************************
********/
Prompt_order()
{
        extern int Sort_type, Snum_vals;
        extern char Sort_vals[][KEYLEN];
```

```
        char input[KEYLEN];
        register int i;

        printf("What do you want to sort on:  ");
        printf("A(uth), S(ubj), C(c), T(ext), D(ate) or Q(uit) ?\n>
        ");
        scanf("%s",input);

        if (input[0] == 'q' || input[0] == 'Q')
                return(-1);

        if (input[0] == 'd' || input[0] == 'D')
        {
                Sort_type = DATE;
                /* set this up only so the List option will show
                'date' */
                strcpy(Sort_vals[0], "date");
                Snum_vals = 1;
        }
        else
        {
                switch (input[0])
                {
                        case 'a':
                        case 'A':
                                Sort_type = AUTHOR;
                                break;
                        case 's':
                        case 'S':
                                Sort_type = SUBJECT;
                                break;
                        case 'c':
                        case 'C':
                                Sort_type = CC;
                                break;
                        case 't':
                        case 'T':
                                Sort_type = TEXT;
                                break;
                        default:
                                printf("Unknown key\n");
                                return(-1);
                                break;
                }

                printf("What do you want to be shown first?\n");
                printf("Use (;) to quit > ");
                for (i=0; i<MAXKEYS; i++)
                {
                        scanf("%s",Sort_vals[i]);
                        if (Sort_vals[i][0] == ';')
                        {
                                Sort_vals[i][0] = ' ';
                                break;
                        }
```

```
                        if (i != MAXKEYS-1)
                                printf("next? > ");
                }
                Snum_vals = i;
                printf("sorting on %d values\n", i);
                fflush(stdout);
        }
        return(0);
}
```

```
#include        <ctype.h>
#include        <stdio.h>
#include        "imail.h"

##char Isubj[SUBJLEN+1];
##char Iauth[AUTHLEN+1];
##long Idate;
##int  Iseqnum;
##char Itext[MAXPARTS][TEXTLEN +TEXTLEN+1];

int Idblen;     /* length of the input stream, according to ingres
*/
##int Totidblen;

##extern char Imrel[];

extern char Lmailfile[];
extern FILE *bugfp;
extern char *copy(), *nextword();
extern int Tilda;


/*****************************************************************
*********

        Parse_mail

        Loop thru each msg in mailfile.
        For each new message, extract the useful info such
        as author, date, subject, and text.
        Store all this in the user's IDB relation.

*****************************************************************
********/
Parse_mail(ldate)
long ldate;     /* last time the IDB was updated for this usr -
logdate */
{
        register int msg_count, i, prev_blank, new_msg;
        char linebuf[80], newauth[AUTHLEN];
        int textpart, linelen, textpos;
        long newdate;
        FILE *lfp;

        if ((lfp = fopen(Lmailfile, "r")) == NULL)
        {
                printf("Parse: Lmailfile fopen failed\n");
                fflush(stdout);
                return(-1);
        }

        textpart = textpos = Idblen = Totidblen = 0;
        msg_count = 0;
        prev_blank = NO;
```

```
        new_msg = NO;
        while (fgets(linebuf, 80, lfp))         /* get one line */
        {
                /* if last char is backslash, put it back */
                if (linebuf[78] == '\\')
                {
                        putc('\\', lfp);
                        linebuf[78] = '\0';
                }
                /* is it the header line? */
                if (ishdr(linebuf, newauth, &newdate))
                {
#ifdef DEBUG7
                        fprintf(bugfp, "linebuf: %s\n", linebuf);
                        fflush(bugfp);
#endif
                        /* have a new message */
                        /* write prev msg to IDB  if need to process
                        another*/
                        /*(if newdate < ldate, msg will be written
                        at end)*/
                        if (msg_count && (newdate > ldate))
                        {
                                /* write it to idb */
#ifdef DEBUG7
                                fprintf(bugfp, "PARSE: write prev
                                msg ");
#endif
                                write_tup(&textpart, &textpos,
                                "tx");
                        }

                        if (newdate > ldate)
                        {
                                /* it's a new message */
                                new_msg = YES;
                                msg_count++;
                                /* init the new msg buffers */
                                Iseqnum = 0;
                                strinit(Iauth, ' ', AUTHLEN);
                                strinit(Isubj, ' ', SUBJLEN);
                                for (i=0; i<MAXPARTS; i++)
                                    strinit(Itext[i],'\0', TEXTLEN
                                    +TEXTLEN +1);

                                strcpy(Iauth, newauth);
                                Idate = newdate;

                                /* if the prev line wasn't blank,
                                put a
                                        blank line in.  A blank line
                                        starts
                                        every new msg.
                                */
                                if (!prev_blank)
```

```
                                        addline(&textpart, &textpos,
                                                "\n", "hd");
                                addline(&textpart, &textpos,
                                linebuf, "hd");

                                fillhdr(lfp, &textpart, &textpos);
                        } /* now have hdr fields for idb; get rest
                        of text*/
                        else    /* reached msgs already have in idb
                        */
                        {
                                new_msg = NO;
                                continue; /*reached a msg already
                                have in idb*/
                        }
                }
                else  /* not part of the header */
                {
                        if (new_msg)
                                addline(&textpart, &textpos,
                                linebuf, "tx");
                        /* else already have it in idb */
                }
                /* each msg must start with a blank line, remember
                if got 1*/
                if (linebuf[0] == '\n')
                        prev_blank = YES;
                else
                        prev_blank = NO;
        }
#ifdef DEBUG7
        fprintf(bugfp, "read all messages, write last 1\n");
        fflush(bugfp);
#endif
        if (msg_count)
        {
/*zzz
                Itext[textpart][textpos] = '\0';
*/
                write_tup(&textpart, &textpos, "tx");
        }
        fclose(lfp);
}
```

```
/*****************************************************************
*********

        addline

        Add a line of the message to the idb text fields.  Manage
        the
        part number properly and start a new record if the current
        one
        is full.  The entire message uses at least 2 idb records:
        the header
        record(s) and the text record(s).  The 2 are never found in
        the
        same record.

*****************************************************************
********/
addline(textpart, textpos, linebuf, tuptyp)
int *textpart, *textpos;
char *tuptyp;
char *linebuf;
{
        register int i;
        int numcopy, linelen, lpos, numichars, numrchars;

#ifdef DEBUG7
        fprintf(bugfp, "addline: a%sa\n", linebuf);
        fprintf(bugfp, "type %s\n", tuptyp);
#endif
        linelen = strlen(linebuf);

        lpos = 0;

        while (lpos < linelen)  /* do each char in linebuf */
        {
                while (*textpart < MAXPARTS)    /* fill a tuple */
                {
#ifdef DEBUG7
                        fprintf(bugfp, "textpart %d, textpos %d\n",
                                *textpart, *textpos);
#endif
                        while ((Idblen < TEXTLEN) && (lpos <
                        linelen))
                        {

                                copychar(&Itext[*textpart][*textpos]

                                    &linebuf[lpos], &numichars,
                                    &numrchars);
                                lpos ++;
                                *textpos += numrchars;
                                Idblen += numichars;
                                Totidblen += numichars;
```

```
                        }       /* end loop to fill a part */

#ifdef DEBUG7
                        fprintf(bugfp, "aa%saa\n",
                        Itext[*textpart]);
                        fprintf(bugfp, "lpos = %d, linelen = %d\n",
                                lpos, linelen);
                        fprintf(bugfp, "textpos = %d, Totidblen =
                        %d\n",
                                *textpos, Totidblen);
                        fflush(bugfp);
#endif
                        if (lpos >= linelen)    /* no more chars in
                        line */
                                break;
                        /* start a new part */
                        Itext[*textpart][*textpos] = '\0';
                        (*textpart)++;
                        *textpos = 0;
                        if ((Idblen > TEXTLEN) && (*textpart <
                        MAXPARTS))
                        {
                                Itext[*textpart][0] =

                                        Itext[(*textpart)-1][Idblen]

                                (*textpos)++;
                                Idblen = 1;
                                Totidblen++;
                        }
                        else
                                Idblen = 0;
                }       /* end loop to fill a tuple */

                if (lpos >= linelen)    /* no more chars in line */
                        break;
                /* this msg is too long for 1 rcd */
#ifdef DEBUG7
                fprintf(bugfp, "PARSE: hdr: starting cont msg:  ");
                fprintf(bugfp, "PARSE: hdr: write prev part:  ");
                fprintf(bugfp, "textpart = %d\n", *textpart);
                fflush(bugfp);
#endif
                write_tup(textpart, textpos, tuptyp);
                *textpart = 0;
                *textpos = 0;
                if (Idblen > TEXTLEN)
                {
                        Itext[*textpart][0] =
                                Itext[(*textpart)-1][Idblen];
                        (*textpos)++;
                        Idblen = 1;
                        Totidblen++;
                }
                else
```

```
                {
                        Idblen = 0;
                        Totidblen = 0;
                }
        }       /* end loop for each char */

}

/*****************************************************************
*********

        any

        Is ch one of the chars in str?

*****************************************************************
********/
any(ch, str)
        char *str;
{
        register char *f;
        register c;

        f = str;
        c = ch;
        while (*f)
                if (c == *f++)
                        return(1);
        return(0);
}

/*****************************************************************
*********

        copy

        Copy str1 to str2, return pointer to null in str2.

*****************************************************************
********/

char *
copy(str1, str2)
        char *str1, *str2;
{
        register char *s1, *s2;

        s1 = str1;
        s2 = str2;
        while (*s1)
                *s2++ = *s1++;
        *s2 = 0;
        return(s2);
}
```

```
/*****************************************************************
*********

        copychar

        Copy the line from the mail file to the idb buffer (Itext).
        Backslash any special characters and do something with
        control characters.

*****************************************************************
********/
copychar(target, src, numichars, numrchars)
char *target, *src;
int *numichars, *numrchars;     /* chars according to ingres; real
chars */
{
        register int i;
        char *tp, *sp;

        *numichars = 1;
        *numrchars = 1;
        sp = src;
        tp = target;

        /* the following characters have been tested and found OK:
           !@#$%^&()_-+={}'~|:;',.<>/
           backslash, when not in front of a special char, is
           not accepted by ingres.
           The idb field must be totally filled or it will get
           padded with
           blanks.  Backslashes don't count as chars to Ingres, so
           keep
           track of the length as Ingres sees it.  Do this only for
           the
           text part of each tuple.
        */
        switch(*sp)
        {
                case '?':
                case '[':
                case ']':
                case '*':
                case '"':
                        if (*(sp-1) != '\\')
                        {
                                *tp++ = '\\';
                                (*numrchars)++;
                        }
                        *tp = *sp;
                        break;
                case '\n':
                case '\t':
                        /* tabs and newlines are OK */
```

```
                        *tp = *sp;
                        break;
                case '~':
                        /* if user specified, change ~ to ~~ */
                        if (Tilda)
                        {
                                *tp++ = *sp;
                                (*numichars)++;
                                (*numrchars)++;
                        }
                        *tp = *sp;
                        break;
                case '\~L':
                        /* Got a CONTROL L */
                        /* change special ctl chars to ~char */
                        if (Tilda)
                        {
                                *tp++ = '~';
                                *tp = 'L';
                                (*numichars)++;
                                (*numrchars)++;
                        }
                        else
                                *tp = ' ';
                        break;
                case '\0':
                        *tp = *sp;
                        *numichars = 0;
                        *numrchars = 0;
                        break;
                default:
                        /* swallow the other control chars */
                        if (iscntrl(*sp))
                        {
                                *tp = ' ';
                        }
                        else
                        {
                                *tp = *sp;
                        }
                        break;
        }
}
```

```
/*****************************************************************
*********

        fillhdr

        Copy the entire header into Itext.  If there's a subject
        copy into the appropriate Ivar.

        Mail always puts a blank line after the header and before
        the text.

*****************************************************************
********/
fillhdr(lfp, textpart, textpos)
FILE *lfp;              /* fp to user's mail file (the 1 imail
builds)*/
int *textpart, *textpos;     /* Itext part num, char pos in Itext */
{
        char *cp, *dp;
        char linebuf[81];
        int  linelen, i, numichars, numrchars;

        /* loop til get the blank line after hdr info, before the
        text */
        while (fgets(linebuf, 81, lfp) != 0)
        {
                if (linebuf[0] == '\n')
                        break;
                cp = linebuf;
                if (strncmp("Subject: ", cp, 9) == 0)
                {
                        cp += 9;
                        dp = Isubj;
                        i = 0;
                        while (i<SUBJLEN)
                        {
                                copychar(&Isubj[i], cp++,
                                &numichars,
                                        &numrchars);
                                i += numrchars;
                                if (Isubj[i-1] == '\n')
                                {
                                        Isubj[i] = '\0';
                                        break;
                                }
                        }
                        Isubj[SUBJLEN] = '\0';
                }
                if (strncmp("Cc: ", cp, 4) == 0)
                {
                        /* CC line gets a tuple(s) of its own.
                           Write the first part of the header, and
                           start
```

```
                        a new tuple.
                    */
                    write_tup(textpart, textpos, "hd");
                    while (1)
                    {
                        addline(textpart, textpos, linebuf,
                        "cc");
                        if (fgets(linebuf, 81, lfp) == 0)
                        {
                            printf("ERROR in reading
                            messages\n");
                            fflush(stdout);
                            break;
                        }
                        if (linebuf[0] == '\n') /* end of header
                        */
                        {
                            write_tup(textpart, textpos, "cc");
                            break;
                        }
                    }
                    break;  /* no need for 'addline'; have
                    EOHeader */
                }

                /* add the header info to the idb record */
                addline(textpart, textpos, linebuf, "hd");

        }  /* end while loop; found a blank line */

        /* put the blank line in */
        linebuf[1] = '\0';
        addline(textpart, textpos, linebuf, "hd");

        /* hdr info is not mixed with text info; write this record
        to idb */
        write_tup(textpart, textpos, "hd");
}


/*********************************************************************
*********

        isdate  and cmatch

        Test to see if the passed string is a ctime(3) generated
        date string as documented in the manual.  The template
        below is used as the criterion of correctness.
        Also, we check for a possible trailing time zone using
        the auxtype template.

        cmatch

        Check the string to see if it is in a valid date format.
        Match the given string against the given template.
```

```
        Return 1 if they match, 0 if they don't

*****************************************************************
********/

#define L       1               /* A lower case char */
#define S       2               /* A space */
#define D       3               /* A digit */
#define O       4               /* An optional digit or space */
#define C       5               /* A colon */
#define N       6               /* A new line */
#define U       7               /* An upper case char */

extern long cmatch();

char ctypes[] =
{U,L,L,S,U,L,L,S,O,D,S,D,D,C,D,D,C,D,D,S,D,D,D,D,0};
char tmztypes[] =
{U,L,L,S,U,L,L,S,O,D,S,D,D,C,D,D,C,D,D,S,U,U,U,S,D,D,D,D,0};

isdate(datestr, date)
char datestr[];
long *date;
{
        register char *cp;

        cp = datestr;
        if (*date = cmatch(cp, ctypes))
                return(1);
        if (*date = cmatch(cp, tmztypes))
                return(1);
        return(0);
}

char months[][3] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
                    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};
extern int gpair();

long
cmatch(str, temp)
char str[], temp[];
{
        register char *cp, *tp;
        register int c;
        int charcount, i;
        int year, month, day, hour, minute, second;

        charcount = 0;
        day = 0;
        cp = str;
        tp = temp;
        while (*cp != '\0' && *tp != 0) {
                c = *cp;
                switch (*tp++) {
                case L:
```

```
                if (c < 'a' || c > 'z')
                        return(0);
                break;

        case U:
                if (c < 'A' || c > 'Z')
                        return(0);
                if (charcount == 4)
                {
                        for (i=0; i<12; i++)
                                if ((strncmp(cp, months[i],
                                3) ==0))
                                        break;
                        month = ++i;
                }
                break;

        case S:
                if (c != ' ')
                        return(0);
                break;

        case D:
                if (c< '0' || c> '9')
                        return(0);
                switch (charcount)
                {
                        case 9:
                                day = day + (c-'0');
                                break;
                        case 11:
                                hour = gpair(cp);
                                break;
                        case 14:
                                minute = gpair(cp);
                                break;
                        case 17:
                                second = gpair(cp);
                                break;
                        case 22:
                        case 26:
                                year = gpair(cp);
                                break;
                }
                break;

        case O:
                if (c != ' ' && (c< '0' || c> '9'))
                        return(0);
                if (c != ' ')
                        if (charcount == 8)
                                day = (c - '0') * 10;
                break;

        case C:
```

```
                        if (c != ':')
                                return(0);
                        break;

                case N:
                        if (c != '\n')
                                return(0);
                        break;
                }
                charcount++;
                cp++;
        }
        if ((*cp != '\0'  && *cp != '\n') || *tp != 0)
                return(0);
        return((long) cnvtime( year, month, day, hour, minute,
        second));
}

/*****************************************************************
*********

        ishdr

        Headers always start with "From " and then the date string.
        Determine if that's true for the line passed in, if so, fill
        in the author and date before returning.

*****************************************************************
********/
ishdr(linebuf, auth, date)
char *linebuf, *auth;   /* current input line; ptr to (empty) author
*/
long *date;             /* ptr to (empty) date */
{
        char word[80];
        char irawdate[32];
        char *cp, *dp;
        char t_auth[30];
        int t1, t2;

        cp = linebuf;
        if (strncmp ("From ", cp, 5) != 0)
                return(0);

        cp = nextword(cp, word);  /*skip over "from",ret ptr to
        nxtwrd in cp*/
        dp = nextword(cp, auth);  /*get auth, put nxtwrd in dp (tty
        or date)*/
        /* take care of special chars in the author, pass bogus t1
        and t2 */
/*zzz
        copychar(auth, t_auth, &t1, &t2);
*/
        if (strncmp(dp, "tty", 3) == 0)
        {
```

```
                cp = nextword(dp, word);        /* get rid of tty
                str */
                if (cp != NOSTR)
                        strcpy(irawdate, cp);
        }
        else
        {
                if (dp != NOSTR)
                        strcpy(irawdate, dp);
        }

        if (isdate(irawdate, date))   /* it's a header */
        {
                return(1);
        }
        else
        {
                return(0);
        }
}


/*****************************************************************
*********

        nextword

        Collect a liberal (space, tab delimited) word into the word
        buffer
        passed.  Also, return a pointer to the next word following
        that,
        or NOSTR if none follow.

*****************************************************************
********/
char *
nextword(wp, wbuf)
char wp[], wbuf[];   /* ptr to input str, ptr to word you are
looking for */
{
        register char *cp, *cp2;

        if ((cp = wp) == NOSTR) {
                copy("", wbuf);
                return(NOSTR);
        }
        cp2 = wbuf;
        while (!any(*cp, " \t") && *cp != '\0')
        {
                if (*cp == '"')
                {
                        *cp2++ = *cp++;
                        while (*cp != '\0' && *cp != '"')
                                *cp2++ = *cp++;
                        if (*cp == '"')
```

```
                        *cp2++ = *cp++;
                } else
                        *cp2++ = *cp++;
        }
        *cp2 = '\0';
        while (any(*cp, " \t"))
                cp++;
        if (*cp == '\0')
                return(NOSTR);
        return(cp);
}

/*****************************************************************
*********

        strinit

        Fill the string with the char passed in.

*****************************************************************
********/
strinit(str, chr, len)
char *str;      /* str to init */
char chr;       /* char to use */
int len;        /* length of str */
{
        char *cp;
        int i;

        cp = str;
        for(i=0; i<len; i++, cp++)
                *cp = chr;
}

/*****************************************************************
*********

        write_tup

        Make the call to append to the idb.  Init Itext, textpart,
        and
        textpos; bump the sequence number.

*****************************************************************
********/
write_tup(textpart, textpos, ituptyp)
int *textpart, *textpos;
##char *ituptyp;
{
        int i;
#ifdef DEBUG7
        fprintf(bugfp, "%s %D\n", Iauth, Idate);
        fflush(bugfp);
        fprintf(bugfp, "APPEND (write_tup), ituptyp = %s\n",
        ituptyp);
```

```
        fprintf(bugfp, "Idblen = %d\n", Idblen);
        fprintf(bugfp, "Totidblen = %d\n", Totidblen);
        fprintf(bugfp, "textpart = %d\n", *textpart);
        for (i=0; i<MAXPARTS; i++)
        {
                fprintf(bugfp, "ww%sww", Itext[i]);
                fflush(bugfp);
        }
#endif

        /*write to idb */
##      append to Imrel
##              (auth=Iauth, subj=Isubj, tup_ty=ituptyp,
##               seqnum=Iseqnum,
##               tuplen=Totidblen,
##               date=Idate, text0=Itext[0],
##               text1=Itext[1], text2=Itext[2],
##               text3=Itext[3], text4=Itext[4])

        *textpart = 0;
        *textpos = 0;
        Idblen = 0;
        Totidblen = 0;
        Iseqnum++;
        /* init only the text; if the auth and subj need to be
        init'ed,
            it's done after the return.
        */
        for (i=0; i<MAXPARTS; i++)
                strinit(Itext[i], '\0',TEXTLEN +TEXTLEN +1);
}
```

```
/* this is the USRMAIL manager.  Routines that need to manipulate
USRMAIL
        into another format are in here
*/
#include        <stdio.h>
#include        <sys/types.h>
#include        <sys/stat.h>
#include        "imail.h"

##      extern char *Usrname;
##      extern char Imrel[];            /* ingres msg file */

    /* The user's mailfile gets copied to Lmailfile and the orig
       mailfile is removed.  Lmailfile originally stood for
       link-mailfile
       but linking doesn't work, so it's copied instead.
    */

extern char Mailfile[];
extern char Lmailfile[];

extern FILE *bugfp;

/*****************************************************************
*********

        Idb_to_mail

        Called at the end of a session, this copies the remaining
        mail messages back into the user's regular mail file.

        return: 1 - new mail has arrived since start of session
                0 - no new mail

*****************************************************************
********/
Idb_to_mail()
{
##      char itext[MAXPARTS][TEXTLEN+1];
        int i, new_mail;
        int tpart, tpos;
##      int ituplen;
        struct stat mailstat;
        char link[100];
        FILE *umfp;     /* put reconstructed msgs back in the user's
        mailfile*/

        if (stat(Mailfile, &mailstat) == 0)     /* had mail when
        started */
        {
                if (mailstat.st_size == 0)      /* but none came in
                */
                        new_mail = NO;
                else
```

```
                        new_mail = YES;         /* there's new stuff
                        */
        }
        else    /* mailfile not there */
                new_mail = NO;          /* so couldn't have new mail
                */

        umfp = fopen(Mailfile, "a");
##      range of idb is Imrel
##      retrieve(
##              ituplen=idb.tuplen,
##              itext[0]=idb.text0, itext[1]=idb.text1,
##              itext[2]=idb.text2, itext[3]=idb.text3,
##              itext[4]=idb.text4)
##      {
                /* need to get rid of the blanks that ingres put
                   in and null terminate the string.
                */
                tpos = ituplen % TEXTLEN;
                tpart = (ituplen - tpos)/TEXTLEN;
#ifdef DEBUG4
                fprintf(bugfp, "ituplen = %d  ", ituplen);
                fprintf(bugfp, "tpart = %d, tpos = %d", tpart,
                tpos);
                fflush(bugfp);
#endif
                for (i=0; i<tpart; i++)
                {
                        itext[i][TEXTLEN] = '\0';
                        fprintf(umfp,"%s", itext[i]);
                }
                if (tpos)       /* if == 0, there's nothing to write
                */
                {
                        itext[tpart][tpos] = '\0';
                        fprintf(umfp,"%s", itext[tpart]);
                }
##      }
        fflush(umfp);
        return(new_mail);
}
```

```
/*****************************************************************
*********

        New_msgs

*****************************************************************
********/
New_msgs()
{

##      extern int Sort_type, Snum_vals;
##      extern char Sort_vals[][KEYLEN];
##      char name[AUTHLEN+1];
##      long ldate, mdate;
        struct stat mailstat;   /* used to get update time on
        mailfile */
        char link[200]; /*used to make Lmailfile, where msgs are
        read from */
        register success, i;
        int len;
        time_t stattime;
        char c, oldc;
        FILE *mfp, *lmfp, *umfp;


        success = 0;

        /* retrieve idb date and default sort order (so can put new
        msgs
           in with the right key)
        */
##      range of ilog is logrel         /* open log relation */

##      retrieve (name = ilog.usrname, Sort_type = ilog.sorttype,
##                      ldate = ilog.logdate, Sort_vals[0] =
ilog.sval0,
##                      Sort_vals[1] = ilog.sval1, Sort_vals[2] =
ilog.sval2,
##                      Sort_vals[3] = ilog.sval3, Sort_vals[4] =
ilog.sval4,
##                      Snum_vals = ilog.num_svals)
##              where (ilog.usrname = Usrname)
##      {
                len = strlen(name);
                name[len] = NULL;
                for (i=0; i<MAXPARTS; i++)
                        addnull(Sort_vals[i], KEYLEN);
                success++;
##      }


        if (!success)
```

```
        {
                printf("Welcome to imail\n");
                fflush(stdout);
                /* unfortunately, defines cannot be used (for
                sorttype) */
##              append to logrel (
##                      logdate=0, usrname=Usrname, logdate=0,
sorttype=4,
##                      num_svals=0)

                ldate = 0;
                Sort_type = DATE;
                Snum_vals = 0;
                /* also create the idbfile for this user */
##              create Imrel
##                      (auth=c20, subj=c50, tuplen=i2,
##                      date=i4, text0=c110, text1=c110, text2=c110,
##                      text3=c110, text4=c110, tup_ty=c2,
##                      seqnum=i2)

                /* ingres keeps a file around for a week only;
                change the
                   expiration date to december of 1999 and hope
                   that's long
                   enough.  There is a bug in Ingres that prevents
                   sending
                   a variable as an argument for the year.
                   Otherwise
                   the expiration date would have been changed to "a
                   year
                   from now" every time the user runs imail.
                */
##              save Imrel until dec 31 1999
        }

        /* get date on USRMAIL */
        if ((stat(Mailfile, &mailstat)) != 0)
        {
                stattime = 0;
        }
        else
                stattime = mailstat.st_mtime;

        /* compare last_update time with date on USRMAIL */
        if (ldate > stattime)
                return(NO);             /* no new msg; idb is up to
                date */


        /* copy the mail file to a temp file; rm the contents of
        mail file */
        sprintf(link, "cp %s %s", Mailfile, Lmailfile);
        system(link);
        umfp = fopen(Mailfile, "w");
        fclose(umfp);
```

```
        Parse_mail(ldate);

        ldate = time(0);
##      replace ilog (logdate = ldate)
##              where ilog.usrname = Usrname

        return(YES);
}
```

```
#include        <stdio.h>
#include        "imail.h"

##extern char Imrel[];
##extern struct delkeep
##      {
##              char iauth[AUTHLEN+1];
##              long idate;
##      }Dk[];
extern int Numdk;

extern FILE *bugfp;

/*********************************************************************
*********

        Temp_to_idb

        Go from the temporary file to the imail db.  Delete from the
        idb
        those messages that were del'd from within mail.

*********************************************************************
********/
Temp_to_idb(Tfp)
FILE *Tfp;      /* fp to Tempfile of remaining msgs */
{
        register int i;
        char linebuf[81], newauth[AUTHLEN];
        long newdate;
        extern char Tempfile[];
##      int currdk;     /* current del-keep index */

##      range of idb is Imrel
        currdk = 0;

        /* if all the messages were removed from the tempfile, the
        open
           will fail.  If that happens, don't try to do the fgets,
           instead
           just skip down to the part where all the rest of the
           messages
           are removed and this will, in effect, remove all the
           messages
           that had been in the tempfile before they were del'ed by
           user.
        */
        if ((Tfp = fopen(Tempfile, "r")) != NULL)
        {
            while (fgets(linebuf, 81, Tfp))             /* get one
            line */
            {
                /* is it the header line? */
                if (ishdr(linebuf, newauth, &newdate))
```

```
                {
                    /* the messages and the entries in delkeep are
                    in
                        the same order.  If there's and entry in
                        delkeep
                        that doesn't have a corresponding message,
                        del it.
                    */

                    while ((Dk[currdk].idate != newdate) ||
                        strncmp(Dk[currdk].iauth, newauth,
                                strlen(newauth)) != 0)
                    {
#ifdef DEBUG4
                        fprintf(bugfp, "Temp_to_idb: DELETE %s
                        %D\n",
                                Dk[currdk].iauth, Dk[currdk].idate);
                        fflush(bugfp);
#endif
##                      delete idb
##                          where idb.auth = Dk[currdk].iauth and
idb.date = Dk[currdk].idate

                        /* maybe it's the next one */
                        currdk++;
                        if (currdk == Numdk)
                        /* del msg from idb */
                        {
                                break;
                        }
                    }
                    currdk++;
                }
                /* else throw it away; need only the headers */
            }
            fclose(Tfp);
        }
        /* do remainder of messages in delkeep file */
        while (currdk < Numdk)
        {
                /* del msg from idb */
##              delete idb
##                  where idb.auth = Dk[currdk].iauth and idb.date =
Dk[currdk].idate
                currdk++;
        }
}
```

```
#include <sys/time.h>
#include <sys/types.h>
#include <stdio.h>

#define dysize(A) (((A)%4)? 365: 366)

static  int     dmsize[12] =
{
        31,
        28,
        31,
        30,
        31,
        30,
        31,
        31,
        30,
        31,
        30,
        31
};

extern FILE *bugfp;
time_t
cnvtime(year, month, day, hour, minute, second)
{
        register int i;
        extern struct tm *localtime();
        time_t tim;
        struct timeval tp;
        struct timezone tzp;

        if(month<1 || month>12)
                return((time_t)-1);
        if(day<1)
                return((time_t)-1);
        if ((day>dmsize[month-1] && month !=2) || (month==2 &&
        day>29))
                return((time_t)-1);
        /* Not a leap year */
        if (dysize(year)==365 && month==2 && day==29)
                return((time_t)-1);
        if(hour<0 || hour>23)
                return((time_t)-1);
        if(minute<0 || minute>59)
                return((time_t)-1);
        if(second<0 || second>59)
                return((time_t)-1);
        if(year<70 || year>99)
                return((time_t)-1);
        tim = 0;
        year += 1900;
        for(i=1970; i<year; i++)
```

```
                tim += dysize(i);
        /* Leap year */
        if (dysize(year)==366 && month >= 3)
                tim += 1;
        while(--month)
                tim += dmsize[month-1];
        tim += (day-1);
        tim = (tim * 24) + hour;
        tim = (tim * 60) + minute;
        if (gettimeofday(&tp, &tzp) == -1)
                fprintf(bugfp, "gettimeofday failed\n");
        tim += tzp.tz_minuteswest;
        tim *= 60;
        tim += second;
        /* check for daylight savings time */
        if(localtime(&tim)->tm_isdst)
                tim -= 60*60;
        return(tim);
}

time_t
gtime(pt)
register char *pt;
{
        int year, month, day, hour, minute, second;
        extern struct tm *localtime();
        time_t now ;

        month = gpair(pt++);
        pt++;
        day = gpair(pt++);
        pt++;
        hour = gpair(pt++);
        pt++;
        minute = gpair(pt++);
        pt++;
        second = 0;
        if (*pt)
                year = gpair(pt);
        else {
                time(&now);
                year = localtime(&now)->tm_year;
        }
        return(cnvtime(year, month, day, hour, minute, second));
}
```

```
int
gpair(pt)
char *pt;
{
        register int c, d;
        register char *cp;

        cp = pt;
        if(*cp == 0)
                return(-1);
        c = (*cp++ - '0') * 10;
        if (c<0 || c>100)
                return(-1);
        if(*cp == 0)
                return(-1);
        if ((d = *cp++ - '0') < 0 || d > 9)
                return(-1);
        return (c+d);
}
```

```
#include <stdio.h>

extern FILE *bugfp;
rmblanks(cp)
char *cp;
{
        char *dp;

        dp = cp + strlen(cp) -1;

        fprintf(bugfp, "dp = %D  cp = %D  strlen = %d\n", dp, cp,
        strlen(cp));
        while(*dp == ' ' && dp != cp -1)
        {
                *dp-- = '\0';
        }
        fprintf(bugfp, "dp = %D  cp = %D  strlen = %d\n", dp, cp,
        strlen(cp));
        fflush(bugfp);
        return(strlen(cp));
}

addnull(cp, len)
char *cp;
int len;
{
        char *dp;

        dp =cp;
        while (*dp != ' ' && (dp != cp + len))
                dp++;
        *dp = '\0';
}
```

```
imail: main.o mailman.o parse.o temp_idb.o usrman.o gtime.o util.o
lib/libq.a
        cc -o imail main.o  mailman.o parse.o temp_idb.o usrman.o
        gtime.o util.o lib/libq.a /usr/lib/libU77.a
        /usr/lib/libI77.a
main.c: main.q imail.h
        equel -d main.q
main.o: main.c
        cc -c main.c
mailman.c: mailman.q imail.h
        equel -d mailman.q
parse.c: parse.q imail.h
        equel -d parse.q
parse.o: parse.c
        cc -c parse.c
temp_idb.c: temp_idb.q imail.h
        equel -d temp_idb.q
usrman.o: usrman.c
        cc -c usrman.c
```

Appendix D

User's Manual for Imail

CONTENTS

User's Manual for Imail

1. What is Imail?

Imail is an enhancement to the Unix* mail command that offers a variety of functions designed to help you select mail messages that are of importance to you. You may search for messages:

- from a particular author
- with a keyword in the subject
- with a keyword in the text
- with a particular person in the "copy-to" list
- relative to particular date and time
- Combinations of the above.

For each of these options, you may select to view just the headers for the messages or to run mail on the matching messages.

You may also choose to sort the mail messages based on the same criteria listed above for searches. Whenever you call imail, your messages will automatically be arranged in the order you have determined.

Imail may be used in place of the Unix mail command or alternately with it. All of the Unix mail capabilities are available through imail.

---

* UNIX is a registered trademark of AT&T Bell Laboratories

2. Using Imail

2.1 How it works

Imail may be called in much the same way that Unix mail is called, that is, simply enter "imail". Figure 2-1 shows the relationship between imail and mail. All of your mail messages are read out of your mailbox and entered into the imail database (A of Figure 2-1). Regular Unix mail is then run automatically for you on all your messages (B of Figure 2-1). You may execute any mail command you wish, including responding to messages and deleting them. When you are finished processing your messages, quit mail and you will be back in the imail environment (C of Figure 2-1). Here, you may perform any of the imail functions that are listed and explained below. When you quit imail, all of your undeleted messages will be copied back to your Unix mailbox (D of Figure 2-1). You will be notified if you have received new mail.

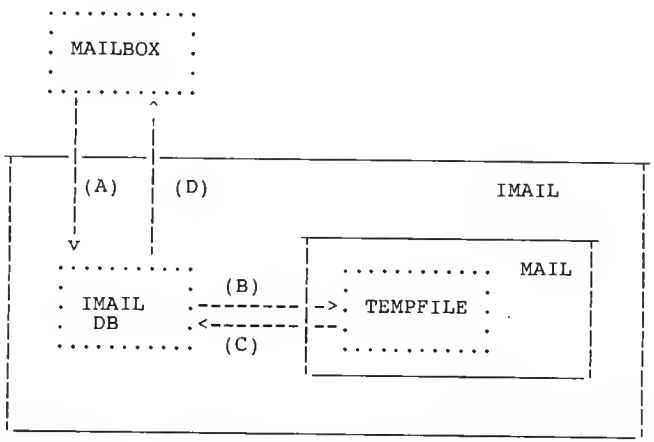


Figure 2-1. Relationship of Imail and Mail Environments

3. The Details of Each Command

When you first enter imail, all the new messages in your mailbox are retrieved and stored in the imail database along with the old messages already there. Imail then sorts the messages according to the order that you set up in advance (more on this later) and automatically proceeds to run mail on these messages. When you have finished processing the messages and quit mail, you are back in the imail environment and are prompted with:

enter: A(ll), T(emp sort), R(eset sort order), L(ist current sort order), S(pecial search), K(ill me), or Q(uit for now)

Any of the options may be selected by entering all or part of the word in either upper or lower case letters. Each of the options is explained below.

3.1 A(ll)

Using the current sorting order, all of the messages will be retrieved from the imail database and passed to the Unix mail command.

3.2 T(emp sort)

Once in the imail environment, you may wish to view your messages in an order other than your default sorting order. This option prompts you for the new sorting order and then passes the messages, in the new order, to Unix mail. This temporary sort order remains in effect until you end the imail session.

When you select this option, you are prompted with:

What do you want to sort on: A(uth), S(ubj), C(c), T(ext), D(ate) or Q(uit)?

If you select "date", there are no more prompts and the messages will be sorted in chronological order. Any of the other choices will result in the prompt:

What do you want to be shown first? Use (;) to quit >

Enter the character string that identifies your first selection criteria. For example, if you had selected to sort on author, you might wish to enter "henry" here.

You need not enter the entire character string, for instance "enry" would also be accepted, but you must match the upper and lower case letters (a search for "Henry" would constitute a search for a different author from "henry"). After you enter the first value, you will be prompted with

next?

You may enter up to five different search strings, each on a new line. If you choose to sort on fewer than five values, simply enter ";" as an entry and the prompting will stop. The messages will then be sorted and passed to the Unix mail command. The key field you requested (author, subject, cc-list, or message text) will be searched for each message to see if the character string you requested as your first key is present. Any messages that are found will be shown first by the mail command. A search is made again for the second key you requested and so forth until there are no more keys. Any remaining messages will appear in chronological order.

3.3 R(eset sort order)

When you first use imail, your mail messages will be sorted by the date of the message. You may choose to override this so that your messages will automatically be sorted according to an order that you specify once. This new order will remain in effect until you specifically override it with another order.

The prompts for the sort order are exactly the same as those listed above for the temporary sort order. Mail will not be automatically invoked on the messages after setting the default sort order. A request to see all the messages will show them in the new order as it will the next time imail is called.

3.4 L(ist current sort order)

This command will list the current values of the sort keys.

3.5 S(pecial search)

It may be useful for you to select just a subset of messages and view the subset independently of the other messages. The special search option allows you to do this. Once you select it, you are given a secondary prompt:

enter A(nd), O(r), (,), or one of the following and then the value: F(rom), C(c), S(ubj), M(sgtext), D(ate) (<> yy mm dd hh mm), Q(uit)

Using the options listed above, you may build a request to select messages from the database. Before going any further, let's take a look at a few sample requests which could be handled:

```
from robin
from robin and subj schedule
(from robin or from virg) and subj schedule
date > 88 06 15 07 00
subj schedule or msgtext schedule
```

The first example requests all the messages from robin. The second is more specific in that it requests only those messages from robin that have the word "schedule" in the subject. The third request still is concerned about only those messages with the word "schedule" in the subject, but these may be from either robin or virg. The fourth example puts no restrictions on the author or subject, but requests those messages that are dated after 7:00 on June 15, 1988. The last example searches for any message that has the word "schedule" in either its subject or in the text of the message itself.

Using the symbols and keywords listed in the prompt, you may construct an unlimited number of requests. Each request must be made on one line, followed by a carriage return. The symbols from the first set in the prompt are used to logically group requests together. The words from the second set form the request and must be followed by a value (for example "from robin"). The date is very specific, it must be followed by a less-than or greater-than sign and by a date in the format [year, month, date, hour, minute] with each of these represented by a two digit string. You need not spell out an entire keyword, just the first letter in either upper or lower case is sufficient. The request format

follows the standard rules of logic and you may use parentheses to logically group restrictions together. Nested parentheses are also permitted.

There are a few limitations on the makeup of a request. The first is that you must exactly match the case of the letters in the mail message. The database request is not capable of mapping lower case letters to upper case, or vice versa. However, it can do partial matches in the same manner discussed in the section on setting up the sort order.

Another limitation is that the actual request to the database may not exceed a certain length. Due to the nature of the request, a search for a string in the message text once it has been translated to the actual database call requires over half of the allowed number of characters. Therefore two restrictions on the message text in the same request are not permitted, although a message text search may be combined with any of the other searches. If you do exceed the limit you will get the error message:

Sorry, the request is too long, try again

and the request will not be sent to the database.

After you enter the restrictions for your database request you will be prompted:

Enter (h) to see headers only, (m) to invoke mail, (q) to quit request:

If you select "m", the messages that match your request will be sent to the Unix mail command and you will be in the mail environment. If you wish to see just the headers and not run mail, select "h". "Q" cancels the request and returns the imail prompt.

3.6 K(ill me)

The kill-me command deletes the imail database relation that contains your mail messages. It also removes any reference to you from the imail database. It does not destroy any messages in your Unix mailbox. If you wish, you may start using imail again at any time.

3.7 Q(uit for now)

When you are finished with a session of imail, use the quit command to exit imail. This restores your Unix mailbox and maintains all of your current messages in the imail database. If you received new mail messages during an imail session, you will be notified when you quit imail.

3.8 Examples

This section contains several examples of using imail to help familiarize you with the kinds of queries that can be made with imail.

```
enter: A(ll), T(emp sort), R(eset sort order),
L(ist current sort order),
S(pecial search), K(ill me), or Q(uit for now)
> s
enter  A(nd), O(r), (, ),
or  one of the following and then the value:
F(rom), C(c), S(ubj), M(sgtext),
D(ate) (<> yy mm dd hh mm), Q(uit)
> subj paper
Enter (h) to see headers only, (m) to invoke mail,
(q) to quit request:   h

   beth          Tue Jun  7 13:45:12 1988   Re:  paper

   beth          Thu Jun  9 07:46:05 1988   Re:  paper

   vanburen      Mon Jun 13 09:27:04 1988   paper

3 messages found.
```

```
enter: A(ll), T(emp sort), R(eset sort order),
L(ist current sort order),
S(pecial search), K(ill me), or Q(uit for now)
> r
What do you want to sort on:  A(uth), S(ubj), C(c),
T(ext), or D(ate)? > auth
What do you want to be shown first?
Use (;) to quit > rich
next? > virg
next? > beth
next? > ;
```

```
sorting on 3 values

enter: A(ll), T(emp sort), R(eset sort order),
L(ist current sort order),
S(pecial search), K(ill me), or Q(uit for now)

> s
enter  A(nd), O(r), (, ),
or  one of the following and then the value:
F(rom), C(c), S(ubj), M(sgtext),
D(ate) (<> yy mm dd hh mm), Q(uit)
> (s paper and f beth) or from rich
Enter (h) to see headers only, (m) to invoke mail,
(q) to quit request:   h


   beth          Tue Jun  7 13:45:12 1988    Re:  paper

   beth          Thu Jun  9 07:46:05 1988    Re:  paper

   rich          Thu Jun  9 09:18:14 1988    New methods

3 messages found.


enter: A(ll), T(emp sort), R(eset sort order),
L(ist current sort order),
S(pecial search), K(ill me), or Q(uit for now)

> s
enter  A(nd), O(r), (, ),
or  one of the following and then the value:
F(rom), C(c), S(ubj), M(sgtext),
D(ate) (<> yy mm dd hh mm), Q(uit)
> m database and date > 88 06 05 00 00
Enter (h) to see headers only, (m) to invoke mail,
(q) to quit request:   m
Mail version 5.2 6/21/85.  Type ? for help.
"/usr/tmp/tminkley": 2 messages
>   1 rich     Thu Jun  9 16:44  20/635 "New methods"
    2 maxwell  Fri Jun 10 10:06  43/2001 "bern2"
& q

enter: A(ll), T(emp sort), R(eset sort order),
L(ist current sort order),
S(pecial search), K(ill me), or Q(uit for now)
> q
```

4. Options to the Imail Command

There are two options to the imail command that may be selected either together or independently of one another.

4.1 Imail -s

When you invoke imail, it normally sorts all your messages and passes them to Unix mail. Sometimes you may wish to skip this step and go right to the imail functions. "Imail -s" still reads all your new messages out of your mailbox and stores them in the imail database but it does not call mail unless you specifically request it.

4.2 Imail -c

Some mail messages may contain the control character "^L" to start a new page when printing. The imail database cannot store this character so it is converted to a blank character. But you may wish to keep track of where the "^L" occurs, so imail gives you the option of converting every occurrence of "^L" to "~L". At the same time, any "~" in the message will be converted to "~~". Thus if you used the "-c" option and the string "hello ~L^L world" were in your mail message, it would be converted to "hello ~~L~L world".

5. A Note of Caution

You may wish to intermix your use of imail and Unix mail. If you choose to do this, it is important that you understand the relationship between the two functions. When imail is called, it stores the new messages in the imail database and empties the Unix mailbox. When you quit imail, all of your undeleted messages are written back to your Unix mailbox. If you now call mail and delete a message, that message is still stored in the imail database. The next time you call imail you will see it as one of your messages and when you quit imail it will be written along with the other messages back into your Unix mailbox. You must delete a message through imail in order to remove it from the imail database.

6. Experienced INGRES Users

Imail uses an INGRES* database to store all mail messages. You may wish to access this database yourself if the queries provided by imail are not sufficient for your needs. For instance, suppose you wanted to create a distribution list for people interested in databases. You could construct your own query to retrieve the author and copy-to list of any message that had the word "database" in it. Using the names you just retrieved, you could make up your distribution list.

There are two ways to access the imail database. The first is directly from the shell using INGRES's Query Language (QUEL), the other is from within a program using Embedded Quel (EQUEL). There are numerous INGRES manuals available to assist you in making a query; this document does not address the syntax or semantics of INGRES queries.

6.1 Introduction to the secrets of imail

The imail database consists of two different kinds of relations. The first contains each mail message for a particular user; there is one relation per user. The format of this relation is given in Table 6-1. Notice that the text portion of each message is divided into five fields. INGRES places a restriction on the length of a field and on the length of a record. The text is broken up into multiple fields and written across several records to compensate for these limitations. The records are kept in order by the sequence number.

The second relation is shown in Table 6-2. This is known as the master relation because it keeps track of the default sort orders for all users and also the Unix time that each user last ran imail. This time is compared with the messages in a user's mailbox in determining if there are any new entries.

---

* INGRES is a product of Relational Technology, Inc. (RTI)

Relation name: im"userID"

field name	type	length	definition
auth	char	20	author of the message
subj	char	50	subject of the message
date	int	4	date of message in Unix time
text0	char	110	first part of message text
text1	char	110	second part of message text
text2	char	110	third part of message text
text3	char	110	fourth part of message text
text4	char	110	fifth part of message text
tuplen	int	2	number of chars in text 0-4
tup_type	char	2	header, CC record, or text valid options: "hd", "cc", "tx"
seqnum	int	2	for multiple part messages

TABLE 6-1. Layout of Relation for Mail Messages for each User

Relation name: logrel

field name	type	length	definition
usrname	char	20	user's login ID
sorttype	int	2	field default sort is based upon option value date 0 author 1 subject 2 cc 3 text 4
sval0	char	20	first key for default sorting
sval1	char	20	second key for default sorting
sval2	char	20	third key for default sorting
sval3	char	20	fourth key for default sorting
sval4	char	20	fifth key for default sorting
num_svals	int	2	number of sort keys given
logdate	int	4	Unix time imail was last run

TABLE 6-2. Layout of Master Relation for Imail

Before you can access the imail database, you need to know the names of the relations. The database itself is called "imaildb" and the master relation is "logrel". The relations that contain the mail messages are given unique names based on a user's login ID. The name of a relation is "im" immediately followed by the ID. Thus if your ID is "henry", your mail messages would be found in the relation called "imhenry".

6.2 Another note of caution

You are given read and write permissions on the relation that contains your own mail messages. It is recommended that you make queries only on this relation to avoid the possibility of damaging it. If you do corrupt this relation, it may also affect the messages in your Unix mailbox the next time you run imail.

Let's take another look at what imail does. The first thing it does is move the contents of your Unix mailbox to "/usr/tmp/lmID" where "ID" is your login name. Then it updates the imail database with any new messages from the lmID file. When you quit imail, it retrieves the messages in the imail database and puts them into your Unix mailbox. Then it removes the lmID file. Note that if the messages in the database were corrupted, your mailbox may also be damaged.

If you recognize this may be a problem, the best thing to do is copy or move your maibox to another file. Then run imail and request the "kill me" command. Move your saved mail file back to your mailbox and it will again be safe to run imail.

7. Think Big

Imail was designed to help you make better use of your mailbox. The queries that it provides can help you organize your messages and search for ones that are of importance at the moment. If there are other queries that would be helpful to you, don't hesitate to construct your own INGRES queries.

The imail database is there for you to query.

Imail: A DBMS for Electronic Mail

by

KATRYN B. INKLEY

B. Phil, Miami University, 1979

AN ABSTRACT OF A REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

An Abstract of the Master's Report

In 1984, CCITT published a set of recommendations that established a framework upon which enhancements to electronic mail services have been based. One of these enhancements is incorporating a database management system with electronic mail. One such system, called "imail", was developed to assist a user in organizing and viewing his mail messages. The main focus of this paper is on the merits and implementation of imail.

Imail is built upon the UNIX(TM) mail system. It allows the user to determine in advance how incoming mail will be sorted and delivered. It allows the user all the functionality of mail as well as the ability to prioritize and sort the messages based upon the author, a member of the "carbon copy" list, a keyword in the subject heading, or the date in the message.

In addition, imail establishes a platform for the user to write his own routines to manipulate his mail messages. This is in keeping with the goal of increasing the flexibility of incorporating a DBMS with electronic mail.